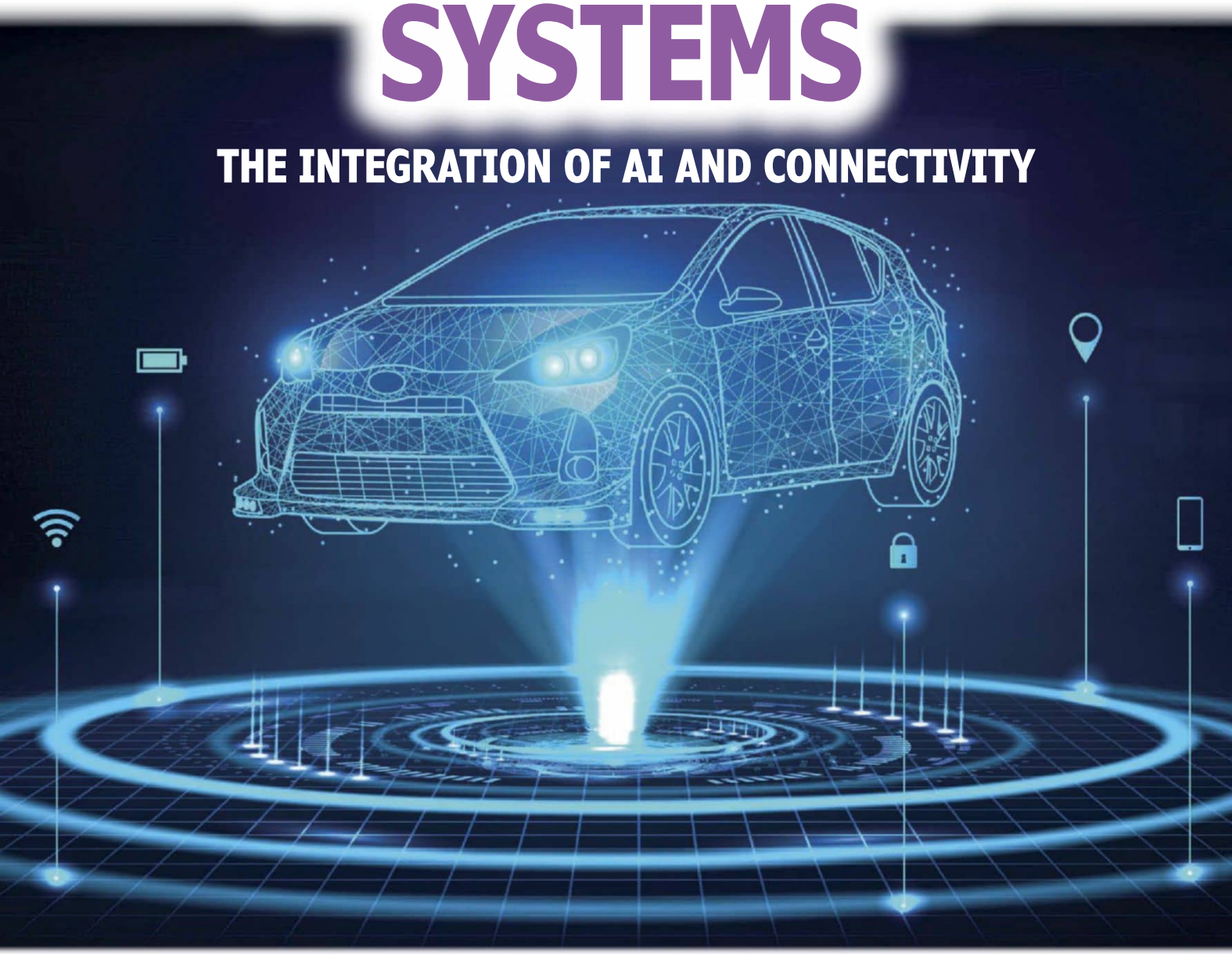
AUTOMOTIVE SYSTEMS
APRIL 2025
ISSUE 417

circuitcellar.com

circuit cellar
Inspiring the Evolution of Embedded Design

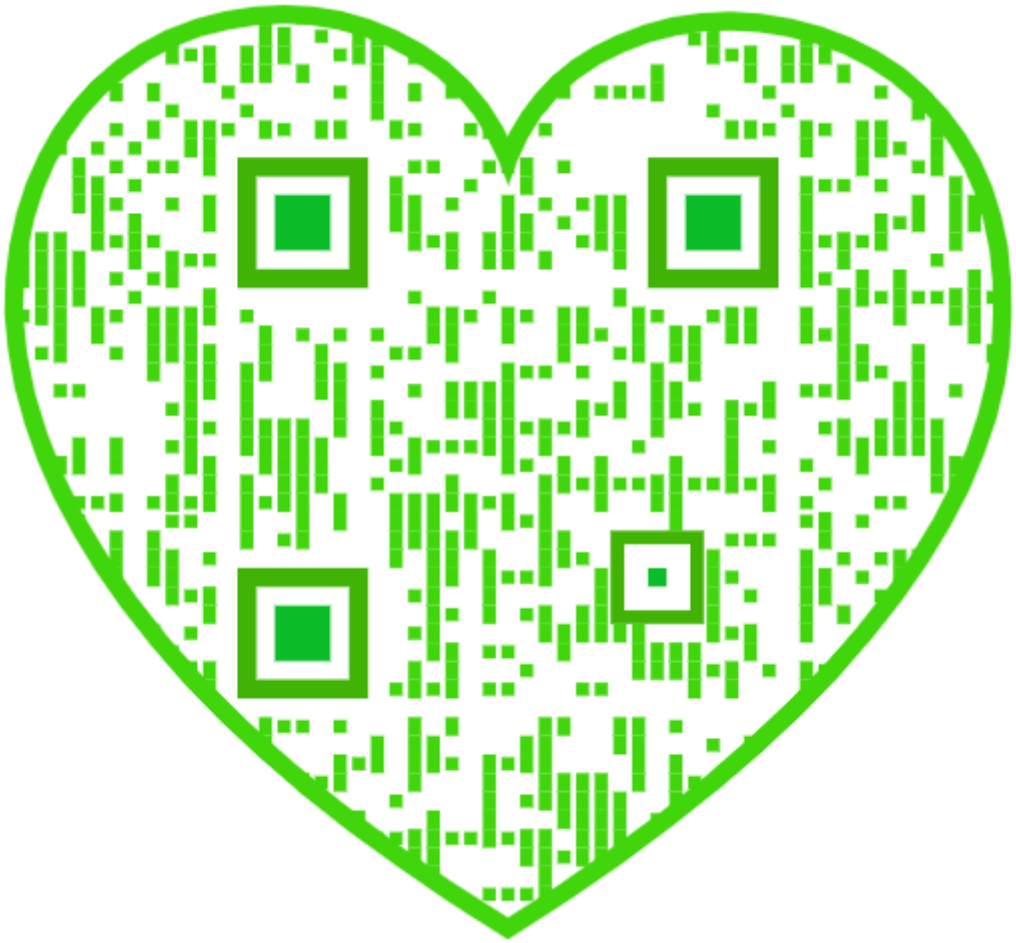
AUTOMOTIVE SYSTEMS

THE INTEGRATION OF AI AND CONNECTIVITY



$9.00US $10.00CAN
04
0 74470 75349 0

32-Bit Microcontrollers | Choosing the Right Embedded OS | ATtiny10 MCU Alternative to Toggle Switch | Three High-Resolution ADCs | Raspberry Pi AI-Driven Smart Home with Intruder Detection | Test-Driven Development in Embedded Systems | Feedback 101 | Introduction to Flutter State Management | The Future of Voice-Enabled AI



Imagine what you will

Discover

400+ ISSUES

CC VAULT

Get the complete *Circuit Cellar* issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Vault archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a *Circuit Cellar* Digital Subscription. Issues appear in searchable PDF format.

cc-webshop.com

Complete archive includes PDFs of all issues in print through date of purchase.

INPUT Voltage

Innovate, Optimize, Secure

Engineers thrive on innovation, constantly seeking to optimize designs. This month's issue of *Circuit Cellar* is packed with insights and practical solutions to empower your projects. From cutting-edge developments in embedded systems and AI to essential techniques for software development and power management, we've curated a collection of articles that address the challenges and opportunities shaping our field. Prepare to dive into a wealth of knowledge designed to inspire, inform, and equip you with the tools you need to excel.

We begin with an exploration into the fundamental principles that govern the behavior of electronic systems. "What Is Feedback and What's It Good For?" delves into the essential role of feedback, both positive and negative, in electrical, software, and system design. You'll gain insights into how feedback affects circuit behavior and stability through examples such as phase-shift oscillators and hysteresis switches.

In the realm of software development, "Adopting Test-Driven Development in Embedded Systems" highlights how Test-Driven Development (TDD) is essential for building reliable and maintainable embedded systems. Jacob Beningo guides you through the TDD microcycle, emphasizing the critical practice of writing tests before implementation to ensure code quality and reduce debugging time.

Next, we tackle the crucial decision of selecting the right operating system for your embedded projects. "Choosing the Right Embedded OS for Your Embedded Solution" helps you determine which type of embedded OS—real time or general purpose (embedded Linux)—is the best for various solutions. Pedro Bertoleti compares real-time operating systems with general-purpose operating systems, detailing their features, differences, and use cases to aid in selecting the optimal OS for your project.

For those working on battery-powered devices, "Soft Power Switch Circuit for Battery-powered Devices" presents a custom solution for controlling power using a simple push button. Michael Hansen's power solution utilizes a tiny CPU, a p-channel field-effect transistor (pFET), and a resistor. The design ensures minimal standby power consumption and compatibility with a wide voltage range, making it an efficient and cost-effective alternative to traditional toggle switches.

Moving into the realm of mobile app development, "Introduction to Flutter State Management" tackles State and State Management in Flutter. Bob Japenga addresses the need for state management in Flutter, distinguishing between ephemeral and app states, and illustrates how to effectively manage widget communication.

Analog-to-digital converters (ADCs) are a cornerstone of many projects, and "A Close Look at High-Resolution ADCs" compares three high-resolution ADCs. Brian Millier offers insights into selecting the right ADC for specific applications, from strain gauge measurements to mains power monitoring, providing key considerations such as resolution, sample rate, input voltage ranges, and single-ended vs. differential configurations.

Finally, we bring it all together with a practical application of AI in home security. "AI-Powered Home Security System, Part 2" Dhairya Parikh continues building an AI-powered home security system using Raspberry Pi 5 and the Hailo AI Kit. The article explains how to add intruder detection and alert functionalities, enabling real-time notifications and remote camera feed access for enhanced home security.

This month's *Circuit Cellar* provides a wide array of articles to keep you at the forefront of engineering innovation. Regardless of your project, we invite you to delve into these articles, expand your knowledge, and apply these insights to your projects.

Caleb Smith

Editor-In-Chief
csmith@circuitcellar.com

Issue 417 April 2025 | ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by:

KCK Media Corp.
PO Box 417, Chase City, VA 23924

Periodical rates paid at Chase City, VA, and additional offices. One-year (12 issues) subscription rate US and possessions $50, Canada $65, Foreign/ ROW $75. All subscription orders payable in US funds only via Visa, MasterCard, international postal money order, or check drawn on US bank.

SUBSCRIPTION MANAGEMENT

Online Account Management: circuitcellar.com/account
Renew | Change Address/E-mail | Check Status

CUSTOMER SERVICE

E-mail: customerservice@circuitcellar.com
Phone: 434.533.0246
Mail: Circuit Cellar, PO Box 417, Chase City, VA 23924

Postmaster: Send address changes to
Circuit Cellar, PO Box 417, Chase City, VA 23924

NEW SUBSCRIPTIONS

circuitcellar.com/subscription

ADVERTISING

Contact: Hugh Heinsohn
Phone: 757-525-3677
Fax: 888-980-1303
E-mail: hheinsohn@circuitcellar.com
Advertising rates and terms available on request.

NEW PRODUCTS

E-mail: product-editor@circuitcellar.com

HEAD OFFICE

KCK Media Corp.
PO Box 417
Chase City, VA 23924
Phone: 434-533-0246

COPYRIGHT NOTICE

Entire contents copyright © 2025 by KCK Media Corp. All rights reserved. Circuit Cellar is a registered trademark of KCK Media Corp. Reproduction of this publication in whole or in part without written consent from KCK Media Corp. is prohibited.

DISCLAIMER

KCK Media Corp. makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors printed in Circuit Cellar®. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, KCK Media Corp. disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar®.

The information provided in Circuit Cellar® by KCK Media Corp. is for educational purposes. KCK Media Corp. makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

© KCK Media Corp. 2025 Printed in the United States

OUR NETWORK

circuit cellar · LinuxGizmos · audioxpress · VOICE COIL · LIS LOUDSPEAKER INDUSTRY SOURCEBOOK

SUPPORTING COMPANIES

Blackmagic Design	9
BusBoard Prototype Systems	51
CCS, Inc.	59
e-con Systems	60
EmbeddedTS	59, C4
Microchip	33
Saelig	55
Siborg Systems, Inc	25
SeaFire Micros, Inc.	C3

NOT A SUPPORTING COMPANY YET?
Contact Hugh Heinsohn
hugh@circuitcellar.com,
Phone: 757-525-3677,
Fax: 888-980-1303
to reserve space in the
next issue of *Circuit Cellar.*

THE TEAM

FOUNDER	Steve Ciarcia
PUBLISHER	KC Prescott
CONTROLLER	Chuck Fellows
EDITOR-IN-CHIEF	Caleb Smith
CONTRIBUTING EDITOR	Brian Millier
CONTENT MANAGER- EU	Jurij Mikeln
SENIOR ASSOCIATE EDITOR	Shannon Becker
TECHNICAL COPY EDITOR	Carol Bower
PROJECT EDITORS	Ken Davidson, David Tweed
MARKETING MANAGER	Tori Zienka
ADVERTISING COORDINATOR	Diana Winsor
ADVERTISING SALES REP.	Hugh Heinsohn

COLUMNISTS
Stuart Ball (Start to Finish)
Jacob Beningo (Modern Embedded Insights)
Bob Japenga (Embedded in Thin Slices)
Brian Millier (Picking Up Mixed Signals)
Colin O'Flynn (Embedded Systems Essentials)

FEATURES

4 **Choosing the Right Embedded OS for Your Embedded Solution**
Real-Time Operating Systems or Embedded Linux
By Pedro Bertoleti

10 **Soft Power Switch Circuit for Battery-Powered Devices**
Alternative to Toggle Switch Uses an ATtiny10 MCU
By Michael Hansen

16 **A Hands-On Look at Three High-Resolution ADCs**
By Brian Millier

26 **Create an AI-Driven Smart Home Security System**
Part 2—Adding Intruder Detection and Alerts
By Dhairya Parikh

34 TECHNOLOGY FEATURE
Automotive Systems
The Integration of AI and Connectivity in Modern Vehicles
By Caleb Smith

38 DATASHEET
32-Bit Microcontrollers
Performance, Integration, and Power Efficiency
By Caleb Smith

COLUMNS

42 Modern Embedded Insights
Adopting Test-Driven Development in Embedded Systems
By Jacob Beningo

46 Start to Finish
Feedback
What Is Feedback and What's It Good For?
By Stuart Ball

52 Embedded in Thin Slices
Introduction to Flutter State Management
By Bob Japenga

61 TECH THE FUTURE
The Future of AI Integration
What It Really Takes for a General-Purpose MCU to Function as a Complete AI SoC
By Mark Rootz

57 : PRODUCT NEWS
60 : TEST YOUR EQ

@editor_cc @circuitcellar · circuitcellar · circuit_cellar

Choosing the Right Embedded OS for Your Embedded Solution

Real-Time Operating Systems or Embedded Linux

By
Pedro Bertoleti

At the heart of every well developed embedded solution, a well tested, validated, and consolidated embedded operating system offers scalability, security, and good overall software quality. In this article, Pedro helps you determine which type of embedded OS—real time or general purpose (embedded Linux)—is the best for various solutions, based on how their respective features, differences, and use cases align with needs.

Embedded systems play a pivotal role in modern technology, forming the backbone of countless devices and applications. Devices ranging from common wearable fitness trackers to complex industrial automation controllers contain embedded systems that require robust and well developed embedded software to function seamlessly.

To offer scalability, security, and good overall software quality, at the heart of a well developed embedded software is a well tested, validated, and consolidated embedded operating system (OS), which manages hardware resources, executes tasks, and ensures system reliability.

Selecting the right embedded OS for a given solution is one of the most critical decisions to be made, since it directly affects the solution's performance, scalability, and long-term maintainability. This article aims to help you make an informed decision, by delving into the core aspects of embedded operating systems, and guiding you through their features, differences, and use cases.

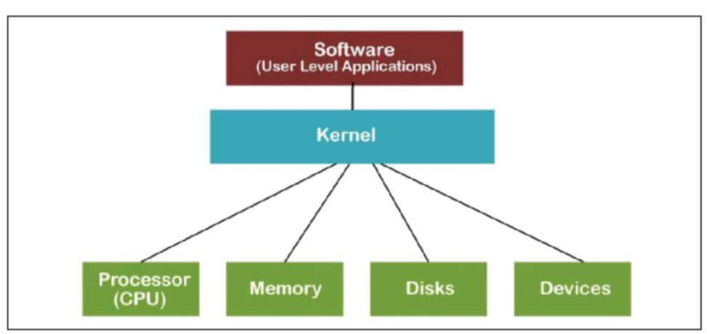


FIGURE 1
The kernel's role in a embedded software solution. Source: Javapoint [1].

OVERVIEW OF AN EMBEDDED OPERATING SYSTEM

An embedded operating system is specialized software designed to run on embedded devices, with its core component known as the "kernel." The kernel is a critical piece of software responsible for managing hardware resources (such as RAM, I/O, and communication interfaces) and overseeing software execution. Its purpose is to ensure secure and efficient task execution, while correctly allocating hardware resources among the software components that make up the system.

An embedded operating system is similar to an orchestra. The conductor (kernel) is responsible for coordinating all the musicians (pieces of software and tasks) to use the musical instruments (system's hardware) as well as possible, to produce a great piece of music (final embedded software solution). The role the kernel plays in an embedded software solution is shown in **Figure 1**.

Thus, embedded operating systems offer developers everything they need to write secure, scalable, and good quality software solutions, which is essential for solutions that are meant to be reliable by design.

Advantages of Embedded Operating Systems: Understanding the advantages of embedded operating systems can help developers to optimize the embedded software development process. The main advantages are summarized here.

1) They simplify the development process by offering pre-built libraries, APIs, and abstraction layers that minimize the need to write low-level code from scratch.
2) They also enable scalability, allowing developers to add new features without major redesigns. Moreover, their optimized scheduling and resource management capabilities ensure high performance, even on hardware with limited resources.
3) Their modular design improves embedded software maintainability, making it easier to debug and update the system as needed.

In essence, an embedded OS serves as the brain of an embedded system, managing its core functions and ensuring that it operates reliably and efficiently, It also makes it easier for developers to build real reliable, scalable, and well developed embedded software.

Categories of Embedded Operating Systems: Embedded operating systems fall into two categories—real-time operating systems and general-purpose operating systems.

A real-time operating system (RTOS) is designed to handle tasks with strict timing and reliability requirements. Its primary goal is to ensure that critical tasks are executed within predictable time constraints, often referred to as "deadlines." The latency in executing tasks should be as brief as possible. This is achieved through features such as preemptive multitasking, priority-based scheduling, and "deterministic" behavior (consistent and predictable guaranteed task execution within a specific timeframe).

RTOSes are commonly used in embedded systems where precise timing is crucial, such as medical devices, automotive systems, and industrial automation. They prioritize real-time performance and minimal latency over other aspects like user interface or extensive multitasking capabilities.

Also, RTOSes are commonly used in solutions that count on a low-tier microcontroller, once its memory/storage/CPU usage footprint is significantly lower than a general-purpose operating systems (GPOS) requires.

Under the hood, RTOSes use two techniques to work as proposed: 1) tasks scheduling (controls which task executes in a given CPU processing time slice); and 2) prioritization (defines which task should be executed first, in case two or more of them are eligible to execute in the same CPU processing time slice).

Figure 2 is a block diagram showing how RTOSes work, from the perspective of task scheduling.

A GPOS is an operating system designed for versatility and usability in a wide range of applications. It is optimized to handle multiple tasks efficiently, but does not guarantee strict timing constraints—meaning latency in executing tasks can be significantly higher than in an RTOS. Examples include Windows, Linux, and macOS, which are used in desktops, servers, and consumer devices.

GPOSes focus on providing robust user interfaces, support for a wide variety of hardware, and the ability to run numerous applications simultaneously. Unlike RTOSes, they are less suitable for applications requiring real-time guarantees, because their scheduling mechanisms prioritize fairness and throughput over timing predictability.

A block diagram of the basic embedded Linux structure, showing how modular it is, is shown in **Figure 3**.

WHEN TO USE AN RTOS OR GPOS

One of the first decisions to make when choosing an embedded operating system is whether to opt foran RTOS or GPOS. This decision depends on the specific needs and constraints of the solution.

As previously noted, RTOSes are designed to handle time-critical applications with deterministic behavior, meaning tasks are executed within guaranteed timeframes, and consequently, with very low latency. In contrast, GPOSes, such as embedded Linux, are better suited for applications that require high support for rich user interfaces, huge support for communication interfaces, high throughput, and can tolerate some variability in timing, such as multimedia processing, IoT gateways, and network equipment.

Another key consideration is the memory and storage footprint of the OS. RTOSes are lightweight and can operate on devices with minimal RAM and storage, making them

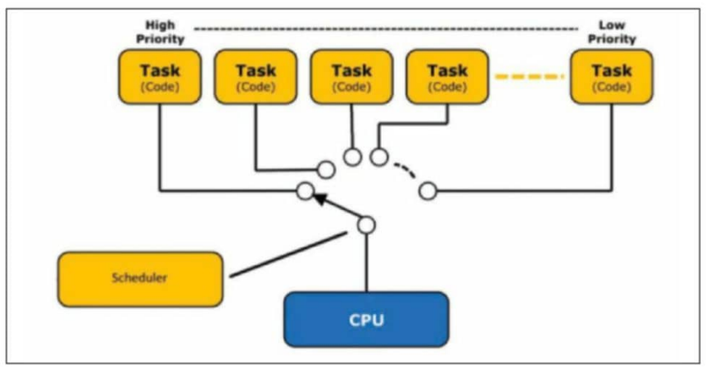


FIGURE 2
Block diagram of tasks time scheduling and prioritization in a real-time operating system. Source: Embarcados [2].

suitable for low-power microcontrollers. Conversely, embedded Linux requires more resources, but offers a wealth of features, including multitasking, virtual memory, and advanced networking capabilities.

Hardware requirements also play a significant role when choosing an embedded OS. RTOSes can run on low-power MCUs with limited processing power, whereas embedded Linux typically requires more capable (and usually more expensive) hardware, such as ARM Cortex-A processors or x86 architectures, which count on much more CPU processing capability, more RAM memory, and memory management unit (MMU) features. Therefore, the more powerful the hardware, the more suitable the use of embedded Linux; and the less powerful the hardware, the more suitable the use of RTOS.

Additionally, the complexity of the embedded software solution influences this choice. RTOSes excel in managing a small number of tasks with strict timing constraints, whereas embedded Linux can handle complex solutions with diverse communication interfaces, graphic user interfaces (GUIs) and threads, and multitasking needs.

Finally, the ecosystem and scalability of the operating system are important factors. RTOSes, such as FreeRTOS and Zephyr, have active communities and are well-suited for MCU-based solutions, from low-tier to advanced. Embedded Linux, with its vast ecosystem of compatible tools and libraries and its well known stability and reliability, even in the most heavy use cases, is the go-to for developers working on resource-rich systems or projects requiring long-term support.

Three commonly used RTOSes, and embedded Linux—a GPOS—are discussed in more detail here.

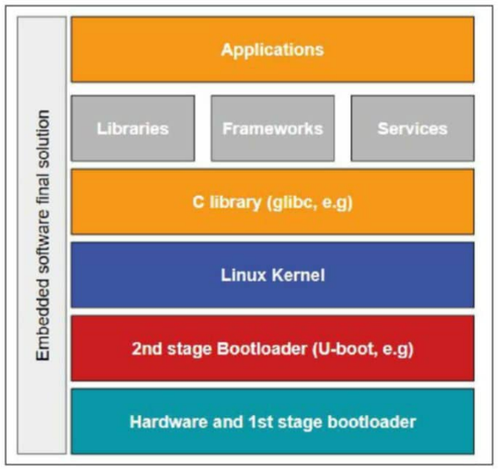


FIGURE 3
Block diagram of embedded Linux structure.

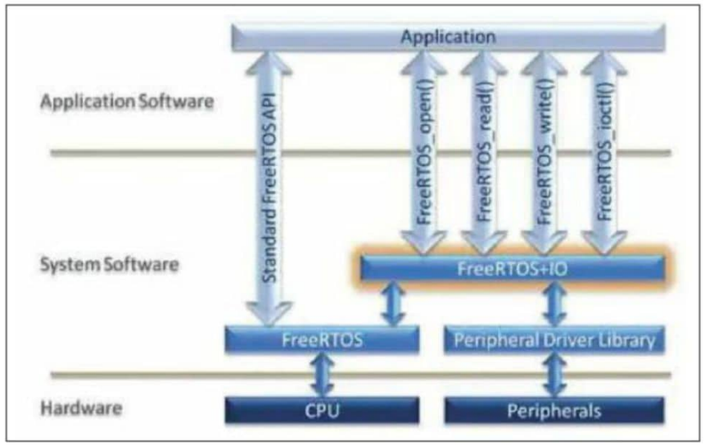


FIGURE 4
Block diagram of FreeRTOS v. 8.1. Source: AspenCore Network (Embedded) [3].

RTOS 1: FREERTOS

A block diagram of FreeRTOS is given in **Figure 4**. FreeRTOS is one of the most widely used open-source RTOSes in the world today, recognized for its simplicity, efficiency, and portability. In terms of software composition, its main components are kernel software components, such as task scheduler, memory management components, and others. FreeRTOS doesn't offer a full-set of tools and drivers to use sensors, actuators, displays, and so on out of the box, but it has a well constructed OS from which to build them.

Its lightweight kernel makes FreeRTOS an excellent choice for resource-constrained systems, and its extensive hardware support ensures compatibility with a wide range of MCUs.

The core of FreeRTOS lies in its preemptive scheduling mechanism, which allows tasks to be executed in order of priority, thereby ensuring that critical tasks are always completed on time, with low latency. Embedded software developers can configure tick rates to achieve precise control over task timing, making it suitable for real-time applications. Additionally, FreeRTOS provides synchronization primitives such as queues, semaphores, and mutexes, enabling efficient communication between tasks.

FreeRTOS is particularly popular today in IoT end-devices, industrial automation, and wearable technology. Its small footprint and easy portability make it an attractive option for projects that require low power consumption and deterministic behavior. Furthermore, its active community and commercial support provide developers with a wealth of resources and tools to expedite development.

RTOS 2: NUTTX

NuttX is a POSIX-compliant RTOS that offers a unique blend of real-time capabilities and Linux-like features, in some cases allowing a whole embedded Linux application to be ported directly to a low-tier microcontroller. This combination makes it a versatile option for developers who need the predictability of an RTOS, but prefer the familiarity of a Linux-style development environment and POSIX APIs.

One of NuttX's standout features is its rich set of drivers and networking stacks, which allow it to support a wide range of hardware and connectivity options, from simple sensors to fancy display and rich user interfaces (using LVGL, for example). Advanced features such as symmetric multiprocessing enable developers to leverage multi-core processors, enhancing performance for more demanding applications. NuttX also maintains a relatively small footprint, making it suitable for devices with moderate resource constraints. **Figure 5** shows a block diagram of NuttX Task Trace Internals.

NuttX is often used in robotics, drones, and smart appliances, where the need for precise timing and advanced functionality intersect. Its scalability and Linux-like environment make it suitable for projects that require both real-time performance and development efficiency.

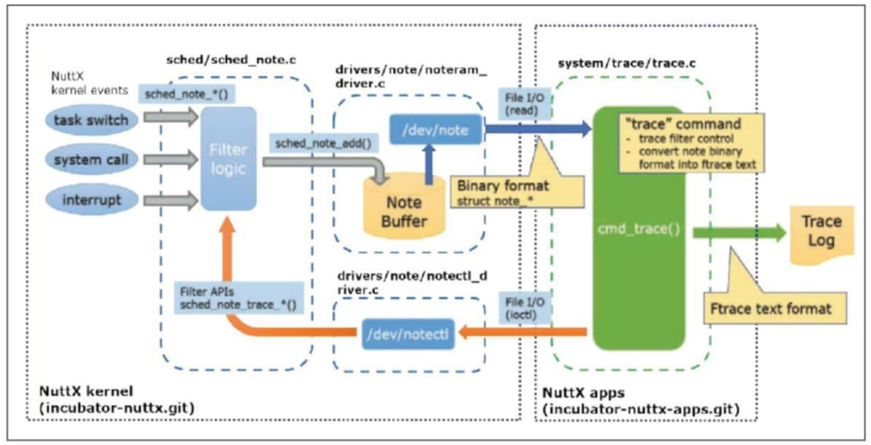


FIGURE 5
Block diagram of NuttX Task Trace Internals. Source: Apache Software Foundation [4].

RTOS 3: ZEPHYR

Zephyr (also known as Zephyr Project), was developed by the Linux Foundation, and is a modern RTOS designed for IoT end-devices and some IoT gateway devices, with connectivity and security in mind. Its modular architecture, strongly based on embedded Linux architecture, allows developers to include only the components they need, thereby minimizing the system's footprint while maintaining flexibility. A simplified block diagram of Zephyr RTOS is shown in **Figure 6**. For more details about Zephyr, please see my article, ""Zephyr RTOS: A Small, Real-Time Operating System for Embedded Devices." in the April, 2024 issue of *Circuit Cellar*.

As with embedded Linux, modularity is a top priority for Zephyr. It also offers an advanced networking stack that supports IoT protocols including MQTT and Bluetooth, making it an excellent choice for connected devices. Zephyr's extensive hardware support and active development community further enhance its appeal.

GPOS: EMBEDDED LINUX

Embedded Linux is a scaled-down version of the Linux kernel that is tailored for embedded systems. It provides the power and flexibility of Linux in a form that can be adapted to resource-constrained environments. Along with this kernel is a set of tools, libraries, and frameworks also tailored for embedded systems comprising embedded

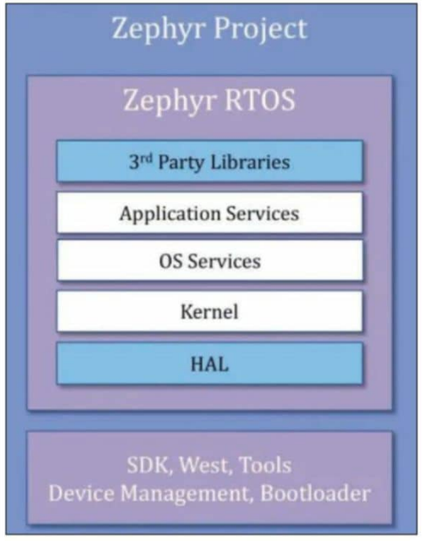


FIGURE 6
A simplified block diagram of the Zephyr RTOS. Source: Bertoleti [5].

Feature	RTOSes: Zephyr, FreeRTOS, NuttX	GPOS: Embedded Linux Distro
Determinism	Deterministic	Non-deterministic
Storage Footprint (order)	Kilobytes	Megabytes
Memory Footprint (order)	Hundreds of kilobytes (in some cases, megabytes)	Hundreds of megabytes (in some cases, gigabytes)
CPU Footprint	Low to moderate. Typically 1 to 2 CPU cores are required, using frequencies having order in hundreds of MHz.	High-performance CPUs required. Typically 4 or more CPU cores are required, using frequencies having order in some GHz.
Hardware Requirements	Low to moderate (low-performance CPUs can be used)	High-performance CPUs and MMU required
Ecosystem	Growing	Extensive
Typical Use Cases	IoT, robotics, real-time control, and critical time-based systems	Multimedia playback, IoT gateways, and rich Graphic User Interfaces

TABLE 1
Comparison of features in real-time operating systems and embedded Linux.

Linux distros. The result is an embedded Linux distro that requires significantly lower memory/CPU processing/storage footprints, when compared to common desktop and/or server Linux distros.

Like RTOSes, embedded Linux is supported for a wide set of SoCs and SiPs, making it possible to develop embedded Linux-based solutions with the huge variety of hardware available today. Also, it's common for SoCs and SiPs to offer the Board Support Package and examples of full Linux distros, which makes possible a smooth integration of embedded Linux and hardware.

One of the key strengths of embedded Linux is its ability to handle multitasking and complex applications and resources. With support for virtual memory, advanced GUIs, and multimedia processing, it is ideal for applications that require a high degree of sophistication. Additionally, its extensive library and toolchain ecosystem offer unparalleled flexibility for developers working with local and network communication interfaces.

Embedded Linux usage is vast, from network equipment, and consumer electronics to automotive multimedia systems and industrial solutions. Its robust security features and high reliability, scalability, and extensive community support make it a preferred choice for developers working on resource-rich systems, with the guarantee of using a well tested and validated embedded operating system.

COMPARISON OF RTOS AND EMBEDDED LINUX USE CASES AND FOOTPRINTS

Table 1 summarizes the key differences between RTOSes and embedded Linux, and highlights their applicable use cases.

CONCLUSION

Choosing the right embedded operating system is a critical decision that affects every aspect of an embedded software solution—from development efficiency to system performance. Real-time operating systems such as FreeRTOS, NuttX, and Zephyr excel in applications with stringent timing and resource constraints, offering lightweight and deterministic behavior. In contrast, embedded Linux provides a powerful and flexible platform for resource-rich systems with complex requirements, such as handling multimedia and complex communication interfaces. By carefully evaluating your project's technical demands, resource constraints, and scalability needs, you can select an embedded OS that ensures success and future-proofing for your application.

ABOUT THE AUTHOR

Pedro Bertoleti is a Brazilian embedded systems engineer and publisher on www.embarcados. com.br, a Brazilian embedded systems website called Embarcados. He has strong experience in development with PIC, ESP32, ARM Cortex M0 and M3 and Rabbit RCM22xx microcontrollers. Pedro also has strong knowledge and experience in IoT projects and Embedded Linux Embedded Systems development.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] to [5] as marked in the article can be found there.

RESOURCES

Apache | www.apache.org

Embarcados | www.embarcados.com.br

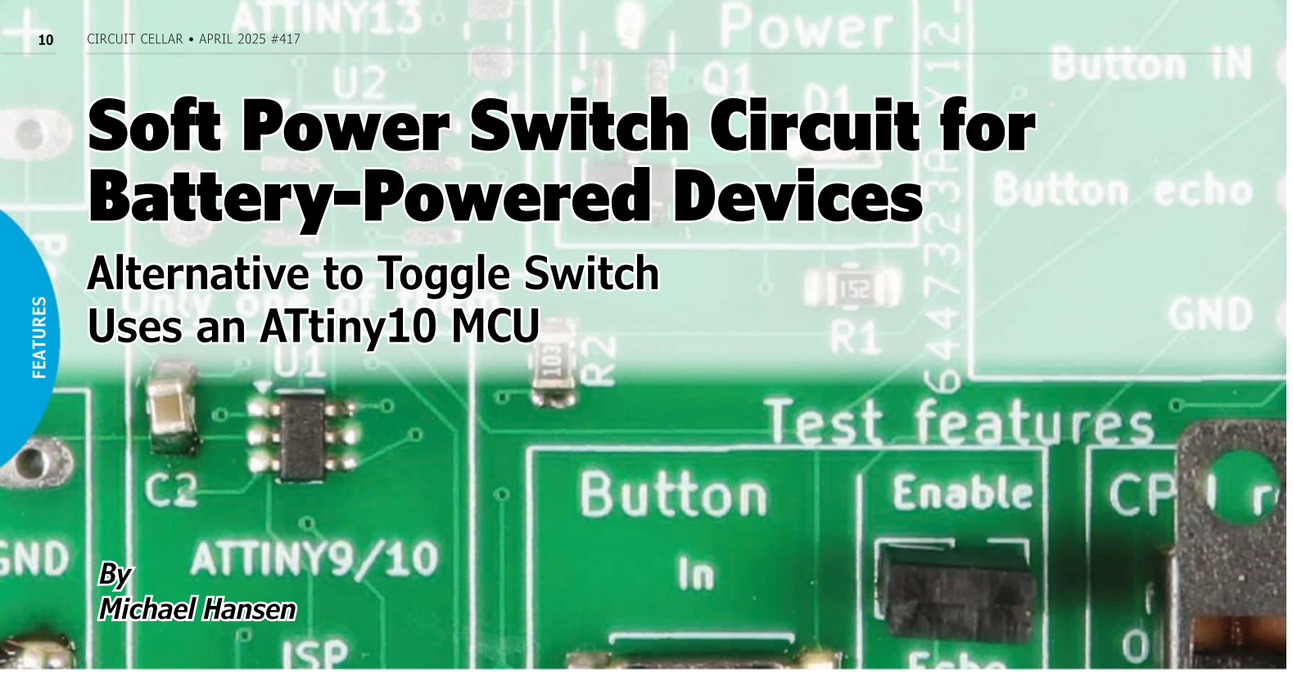
Embedded | www.embedded.com

Javatpoint | www.javatpoint.com

FREE Download Now!
Mac, Windows and Linux
Blackmagicdesign
Edit and color correct using the same software used by Hollywood, for free!
DaVinci Resolve is Hollywood's most popular software! Now it's easy to create feature film quality videos by using professional color correction, editing, audio and visual effects. Because DaVinci Resolve is free, you're not locked into a cloud license so you won't lose your work if you stop paying a monthly fee. There's no monthly fee, no embedded ads and no user tracking.
Editing, Color, Audio and Effects!
DaVinci Resolve is the world's only solution that combines editing, color correction, visual effects, motion graphics and audio post production all in one software tool! You can work faster because you don't have to learn multiple apps or switch software for different tasks. For example, just click the color page for color, or the edit page for editing! It's so incredibly fast!
Professional Editing
DaVinci Resolve is perfect for editing sales or training videos! The familiar track layout makes it easy to learn, while being powerful enough for professional editors. You also get a library full of hundreds of titles, transitions and effects that you can add and animate! Plus, DaVinci Resolve is used on high end work, so you are learning advanced skills used in TV and film.
Creative Color Correction
DaVinci Resolve's color page is Hollywood's most advanced color corrector and has been used on more feature films and television shows than any other system! It has exciting new features to make it easier to get amazing results, even while learning the more advanced color correction tools. There's PowerWindows™, qualifiers, tracking, advanced HDR grading tools and more!
Designed to Grow With You
DaVinci Resolve is designed for collaboration so as you work on larger jobs you can add users and all work on the same projects, at the same time. You can also expand DaVinci Resolve by adding a range of color control panels that let you create unique looks that are impossible with a mouse and keyboard. There's also edit keyboards and Fairlight audio consoles for sound studios!
DaVinci Resolve 19 ..... Free
DaVinci Resolve Micro Color Panel ..... Only $509
Learn the basics for free then get more creative control with our accessories!
Learn more at www.blackmagicdesign.com
Download free on the DaVinci Resolve website
NO SUBSCRIPTIONS • NO ADS • NO USER TRACKING • NO AI TRAINING

Soft Power Switch Circuit for Battery-Powered Devices

Alternative to Toggle Switch Uses an ATtiny10 MCU



By Michael Hansen

The battery in a battery-powered CPU system typically is activated with a toggle switch. In this article, Michael explains how he overcame the limitations of toggle switches by creating a solution that allows a device to be turned on and off using a simple, momentary push button. Additionally, the CPU system, itself, can control the power, making the solution compact, efficient, and inexpensive to construct.

I frequently design devices such as laser tag guns and lightsabers, which typically include a CPU board with a USB socket, a lithium-ion battery, and a built-in charger. This setup allows the CPU system to be updated via the USB interface, but also serves as the charging connection for the battery. However, using a toggle switch to control the battery introduces some limitations:

- The CPU system cannot turn off the power when the battery is running low.
- The CPU system cannot automatically turn on the battery during charging.
- The toggle switch occupies space and cannot be repurposed by the system CPU.

For years, I searched for a solution to address these issues. While some commercial products include this functionality, I was unable to find a readily available, comprehensive solution. Partial solutions exist for individual problems, but none resolved all the challenges.

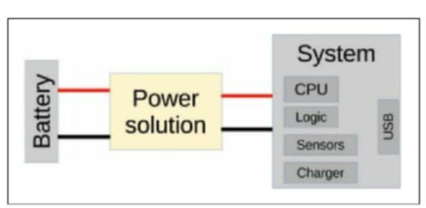


FIGURE 1
Block diagram showing the split between the power solution and the system.

In this article, I describe the custom solution I developed. It effectively addresses the issues mentioned above, while remaining small, efficient, and inexpensive to implement.

FEATURES

Before we dive into the solution, let's look at an outline of the required features. For clarity, everything that needs to be powered on and off—such as the system CPU, charger, and so on—is referred to as the "system." The component that controls power on and off is called the "power solution" (see **Figure 1**). The power solution must provide six features:

Push Button Control: A push button that can be pressed for a few seconds to turn the power to the system on and off. Power off is also used in case the system CPU is stuck and needs a reset.

System-Controlled Power: The system must be able to turn the power on and off programmatically. Power on is used during charging. (The system is powered by the USB but needs access to the battery to charge it.) Power off is used when the battery is running low, to protect the battery; but it can also be used if the device is idle for a certain amount of time.

Shared Push Button: The push button's state must be readable by the system for other purposes, such as configuration. This enables the power solution and system to share the same physical button, saving space and components.

Minimal Standby Power: In off mode, the power solution must not drain the battery, keeping the standby power for the power solution alone lower than the battery's self-discharge rate.

Battery Compatibility: The power solution must operate directly from the battery without any regulator, since regulators will drain the battery. It must support a wide voltage range to accommodate the typical Li-ion battery range of 2.5V to 4.2V.

Compact and Cost Effective Design: The power solution must have a small footprint, making it easy to integrate into compact designs. Additionally, it must be inexpensive, ensuring its cost is negligible compared to the overall system cost.

DESIGN

The power solution utilizes a tiny CPU, a p-channel field-effect transistor (pFET), a resistor, and a push button, as depicted in **Figure 2**. The power "switch" that transitions battery power to the system is a pFET, which is managed by the power CPU. It is important to select a pFET capable of handling the system's power requirements. In this example, the SI2305, a SOT23-3 device capable of handling up to 4A, has been chosen, making it suitable for most battery-powered systems.

The push button connects directly to the power CPU, using an internal pull-up resistor. However, the system must also be able to monitor the button's state. This introduces a challenge, because the button then must span two different power domains. The power CPU operates in the battery power domain, with logic levels up to 4.2V, while the system's power domain typically operates at 3.3V or lower. Directly connecting the button to a system CPU I/O pin would force the pin to 4.2V, potentially damaging it.

To address this, the button state is echoed on another pin of the power CPU. When the button is pressed, the power CPU outputs LOW; when the button is released, the pin is tri-stated. The system CPU must use an internal pull-up resistor on its input pin, thereby controlling the HIGH logic level. This approach ensures the power solution adapts to the system CPU's power domain (see **Figure 3**).

Additionally, the system CPU can manage power on and off using the CPU request pin. This pin requires an external pull-down resistor to ensure a defined logic LOW level when the system CPU is powered down.

STATE MACHINE

The core functionality of the power CPU firmware is implemented using a state machine, as illustrated in **Figure 4**.

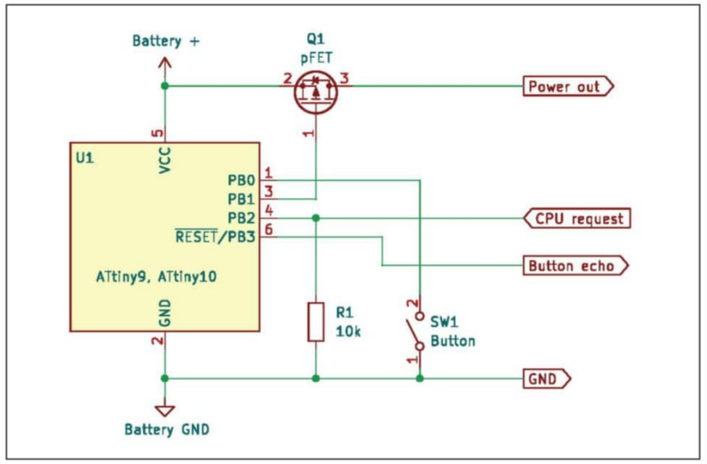


FIGURE 2
Schematic of the power solution. "CPU request" is a simple logic level signal.

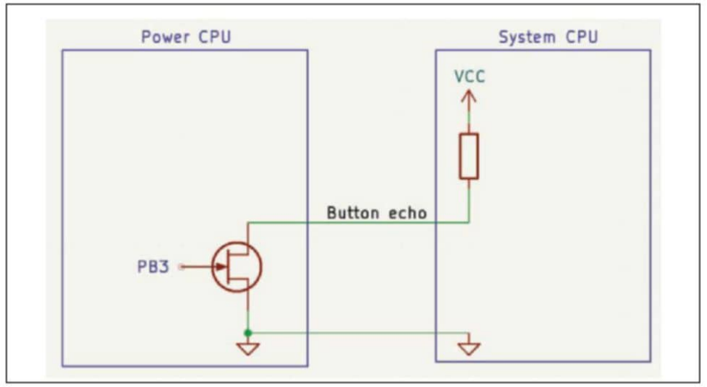


FIGURE 3
The button echo logic level is converted to the system CPU logic level.

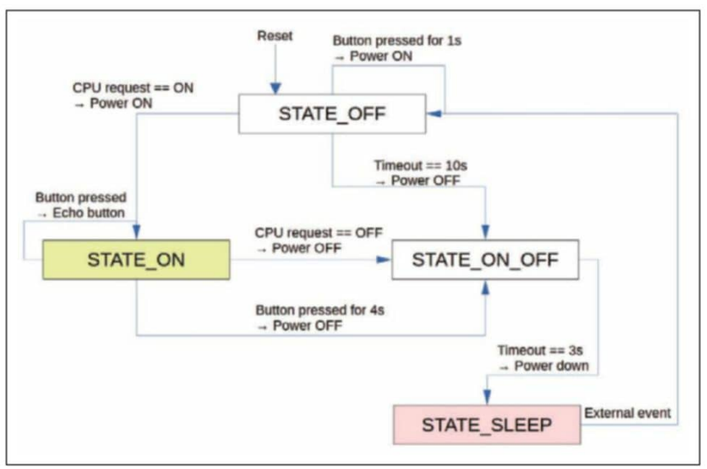


FIGURE 4
The core functionality of the power CPU firmware is implemented using a state machine.

FEATURES

When the power CPU exits reset, it begins in STATE_OFF. If no activity occurs, the CPU transitions to "STATE_ON_OFF" after a 10-second timeout, and then to "STATE_SLEEP" mode after an additional 3 seconds.

When the button is pressed for more than 1 second, the power CPU wakes up, switches to STATE_OFF, and turns on the system power. It remains in this state until the system CPU sets the CPU request signal to HIGH. This transition is necessary, because the system CPU's ability to set the request signal HIGH is the mechanism for later requesting power off. Once the request signal is HIGH, the power CPU transitions to STATE_ON and remains there until a power-off request is made.

If the button is pressed for more than 4 seconds or the system CPU requests power off, the power CPU turns off the system power and moves to STATE_ON_OFF. In this state, it waits for 3 seconds to ensure the system has powered down fully before entering STATE_SLEEP. The 3-second delay prevents power-down noise on the system CPU request pin from causing unintended behavior.

When the system CPU requests power on for activities such as charging, the power CPU wakes up, connects the battery to the system, and transitions to STATE_ON.

While in STATE_ON, the power CPU echoes the button state to the system CPU. This allows the system CPU to read the button state and use it for configuration or other purposes.

This state machine design ensures efficient power management and reliable communication between the power and system CPUs.

POWER CPU FIRMWARE

This solution is designed with an ATtiny low-power microcontroller with the smallest footprint. There exist four nearly identical models: ATtiny4, ATtiny5, ATtiny9, and ATtiny10. All these devices feature four I/O pins and 32 bytes of RAM, and differ only in flash memory capacity. ATtiny4/5 have 512 bytes, whereas ATtiny9/10 offer 1,024 bytes. They are available in two package options, including the SOT23-6 (3x3mm), which can be hand soldered. These devices operate within a voltage range of 1.8V to 5.5V, meeting the project's requirements. For low-quantity purchases, all models are priced similarly so for development, I chose the ATtiny10 due to its larger flash size, though the code is compatible across all four devices.

The code is written in bare-metal C, with no room for unnecessary overhead. This approach requires manually programming all aspects, including CPU initialization, timer configuration, and more. The initialization code is given in **Listing 1.** Instead of providing a detailed walk-through of the code, I'll highlight key challenges addressed during development.

During initialization, all I/O pins are configured, and the state machine is implemented in function PollState().

A hardware timer (Timer 0) is configured as a counter running at 16 counts per millisecond. A custom delay function uses this counter to measure delays in milliseconds, supporting the state machine's timing requirements.

When the system is powered off, the CPU enters power-down mode, consuming approximately 100nA as per the datasheet [1]. Achieving this minimal power usage is one of the primary challenges. By default, power-down mode consumes more power due to unnecessary subsystems still running. These are disabled during CPU initialization to meet the ultra-low power requirement.

The power button connects to a CPU pin with an internal pull-up resistor, eliminating the need for an external resistor. The CPU is programmed to wake up from power-down mode on a pin-change event on either the button pin or the CPU request pin. However, this generates an unwanted interrupt, which is handled by a dummy interrupt handler.

Upon waking, the CPU performs a reset to clear any potential memory leaks or errors. This ensures the system always starts in a clean state, preventing long-term failures. The reset is triggered by enabling the watchdog timer, which automatically resets the CPU after a maximum of 16ms.

```c
int main(void)
{
  // Initialize AVR
  RSTFLR = 0;  // Disable watchdog (save power)
  CCP = 0xD8;  // Special value to enable WD changes
  WDTCSR = 0;  // Disable watchdog

  ACSR = BIT(ACD); // Disable Analog comparator (save power)
  PUEB |= BIT(GPI_BUTTON);  // Button pull-up
  PORTB |= BIT(GPO_PWR);    // PWR off
  DDRB |= BIT(GPO_PWR);     // PWR output

  CCP = 0xD8;  // Special value to enable clock change
  CLKPSR = 0;  // Prescale = 1
  CCP = 0xD8;  // Special value to enable clock change
  CLKMSR = 1;  // Internal 128kHz clock

  /* Timer 0
    Div 8
      128 kHz/8 -> 32 kHz (16 clocks pr. 1 ms)   */
  TCCR0B = 2;

  while(1)
  {
    PollState();
    WaitMs(POLL_DELAY_MS);
  }
}
```

LISTING 1
The ATtiny4/5/9/10 CPU initialization code.

ISP PROGRAMMING

The device needs to be programmed before it can be used, and in "normal" cases, this would be done by using a programmer with a socket that supports the device.

However, due to the device's small size, finding a suitable socket can be challenging. It is possible to buy a programmer and then buy a special socket adaptor that fits the selected device, but it gets expensive. Instead, I have chosen to use a programmer called an ISP (in-system programming) that can program the device on the PCB, I used an AVRISP MKII, which I already had. There exist others (newer) that also will work, and in some cases, have more features. There are, however, a few challenges when using an ISP approach. First, the ISP needs a 6-pin connector to the PCB to program the device; it can be a simple edge connector or a physical pin row. I used the

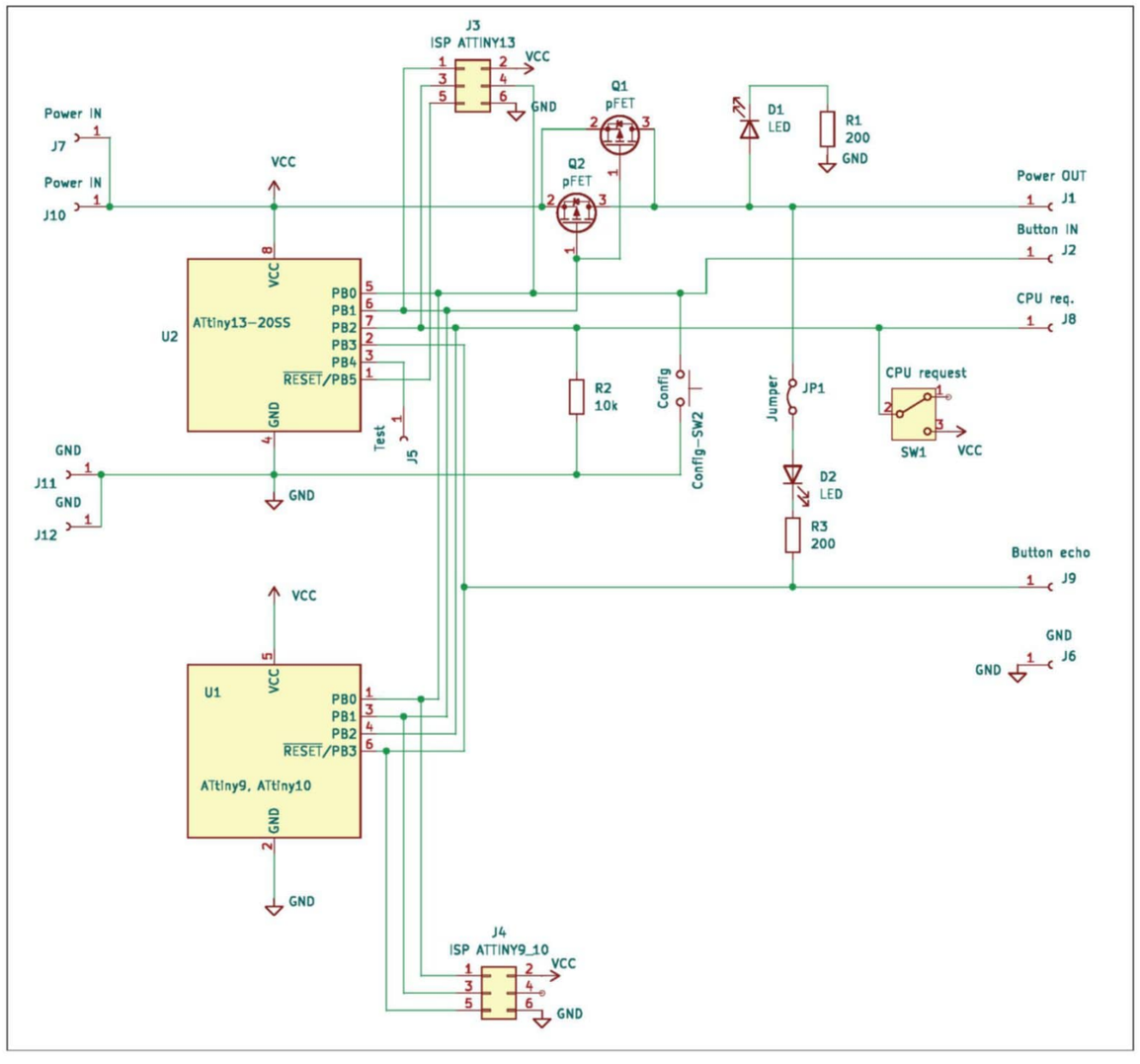


FIGURE 5
Schematic of the evaluation board. Only one of the two ATtiny CPU's can be mounted at one time.

edge connector for the evaluation board that is described in the next section.

Second, The programming voltage needs to be 5V. This will typically require that the power CPU is programmed before the system is connected (soldered); otherwise, the system will get 5V on the shared I/O pins, which it most likely could not survive. If the board is hand soldered, this is not a big deal. Just solder the power CPU first, then program it, and then solder the rest of the components. It is not expected that the power CPU should ever be updated.

The power CPU has only four I/O pins, all of which are used. During programming, the programmer needs to reset the CPU, which conflicts with one of these pins. To resolve this, program the CPU first, and then program the reset fuse to disable the reset functionality of the reset pin. This converts the reset pin into a regular I/O pin. However, disabling the reset pin means the CPU can no longer be reprogrammed. Therefore, test the code thoroughly before setting this fuse. Although some programmers can reverse this fuse, it is often cheaper to replace the CPU and solder a new one.

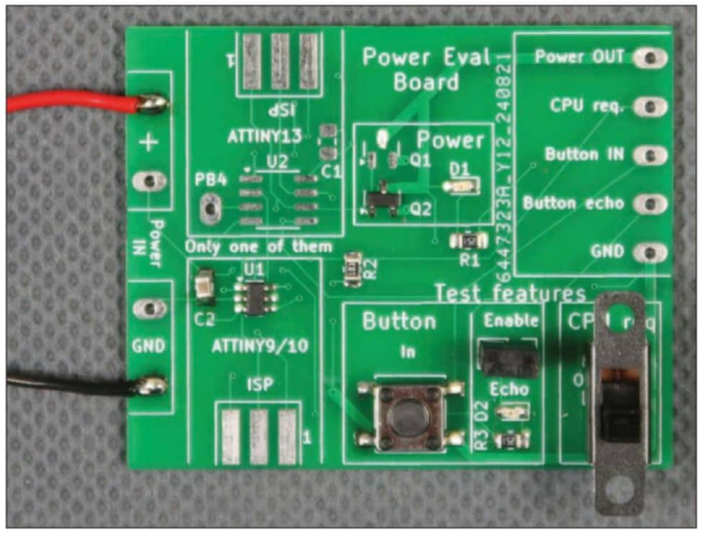


FIGURE 6
Evaluations board with an ATtiny10 mounted. The board supports both an ATtiny4/5/9/10 and ATtiny13. For the ATtiny10 (mounted), it is the ISP edge connector at the bottom that is used.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] to [2] as marked in the article can be found there.

RESOURCES

Microchip | www.microchip.com

The code is compiled using Microchip Studio, because it supports the AVRISP MKII programmer. If a newer programmer is used, Microchip offers additional software options. The summary of the programming workflow includes.

- Select the actual CPU variant.
- Build everything (compile and link everything together).
- Attach the programmer to the power CPU and ensure the CPU is powered with 5V, since the programmer does not supply power.
- Program the file to the CPU's flash memory. Then, test that everything is working as expected.
- Once the code is verified, program the reset fuse (RSTDISBL) to convert the reset pin into an I/O pin (used for button echo functionality). Keep in mind that this step disables further programming of the device.
- Disconnect the programmer from the power CPU. Now the power solution is ready for use.

EVALUATION BOARD

Before integrating the power solution into a final product, it must first be developed and thoroughly tested. To achieve this, I needed a platform tailored for development and testing that would allow all features to be evaluated while still permitting CPU updates.

The ATtiny10, described earlier, is unsuitable for this purpose, because its button echo feature cannot be tested without permanently disabling further programming. Instead, I opted to use an ATtiny13 for development and testing. The ATtiny13 is similar to the ATtiny10 but offers six I/O pins, enabling comprehensive testing of all features without the need to disable programming.

When the test is complete, it is also useful to test the final implementation on the ATtiny10, and specifically test that the special fuse programming is working. For that purpose, I built an evaluation board, that supports ATtiny4/5/9/10 and ATtiny13 devices and has hardware to test all features. It also has pads to connect a full system to it, for a more realistic test.

The board schematic is shown in **Figure 5.** It has support for the power CPU with two different footprints and two different edge connectors for programming the CPUs. It is only possible to mount one of the CPUs on each board, which can then be programmed using the associated edge connector. It is possible to mount two pFET (in parallel), just in case more current is needed, though one is typically enough.

The board has three special test features:

- An LED (D1) that indicates if the power to the system is on.
- A button and an LED (D2) on the button echo signal, to see that the button echo feature is working.
- A switch to emulate the CPU request signal.

The evaluation board is shown in **Figure 6**. The plot files for PCB production can be found on the *Circuit Cellar* Article Materials and Resources webpage.

I began by writing the code for the ATtiny13 and tested all the features to ensure they worked as intended. The button echo function performed so quickly that any slight delay, caused by reading the button state and then echoing it, was imperceptible.

Next, I measured the power consumption during power-down mode, and fine-tuned the CPU parameters to achieve the datasheet-specified power consumption. The final solution with the ATtiny13 consumed approximately 300nA, which is the lowest possible value according to the datasheet [2].

Afterward, I ported the code to the ATtiny10. During this process, I discovered that many of the configuration registers were incompatible with the ATtiny10, despite most other family variants being compatible. I rewrote the configuration and other special registers to match the ATtiny10's requirements. I then re-optimized the power consumption in power-down mode, achieving approximately 100nA, the lowest possible value as specified in the ATtiny10 datasheet [1].

NEXT STEPS

All six required features outlined at the start were met. However, the power CPUs offer additional built-in features such as the ADC that could be utilized. Although the ATtiny10 is out of spare I/O pins, a spare pin could be freed up if the button echo feature is not required or if the ATtiny13 version is used.

This spare pin could be used to monitor the battery voltage and cut power to the system if the battery runs low. Implementing this feature would require only two external resistors to divide the voltage, which would then be connected to the spare I/O pin. The CPU uses its ADC to measure the battery voltage and take appropriate action if needed. To avoid unnecessary power consumption during power-down mode, the resistors should be placed after the pFET.

In my designs, I typically integrate voltage monitoring into the system CPU, allowing the battery level to be displayed on an LED or screen. However, this stand-alone approach may be beneficial for others who require simpler monitoring functionality.

CONCLUSION

I have described two implementations of the power solution, both of which consume so little power that their usage is negligible compared to the battery's self-discharge rate. Both solutions share the same feature set, are compact, and cost around $1-$2. For completeness, a schematic of the power solution for the ATtiny4/5/9/10 with the ISP connector are shown in **Figure 7,** which is needed to program the device.

So far, I have tested the power solution on my evaluation board. I have also integrated it into the next version of my laser tag guns, which will undergo testing soon. This integration enhances user-friendliness by eliminating the risk of forgetting to turn the devices off. Additionally, charging is simplified—users only need to plug in a USB power cable ,and don't need the unlogic step of turning on the power switch.

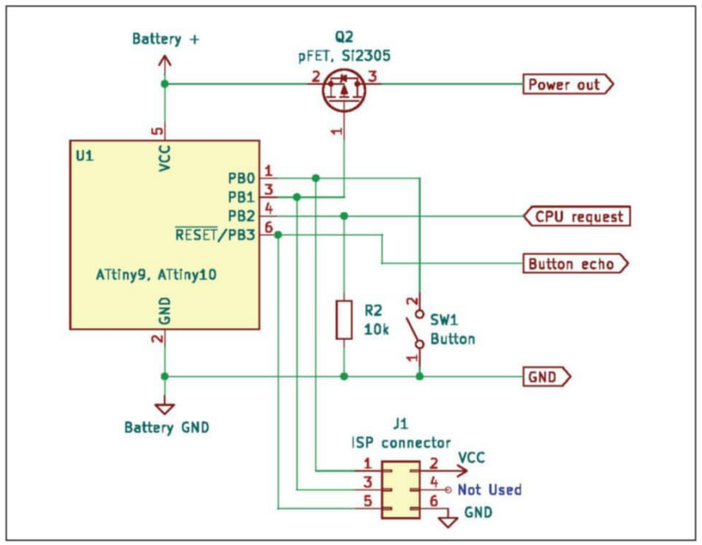


FIGURE 7
Schematic of the power solution for the ATtiny4/5/9/10 with the ISP connector.

ABOUT THE AUTHOR

Michael Hansen holds an M.Sc.Eng degree and has worked as an embedded software developer for 30 years. He now works as a volunteer and helps kids with IT projects. Michael has a small workshop where he builds IT devices such as laser tag guns, lightsabers, RC toys, and smart home devices.

Email: maker@rclab.dk

FEATURES

A Hands-On Look at Three High-Resolution ADCs

In this article Brian takes a close look at three different high-resolution ADCs and compares their features, strengths, and more. Two of the three devices have shown up in his past projects and he anticipates using the third in the future.

By
Brian Millier

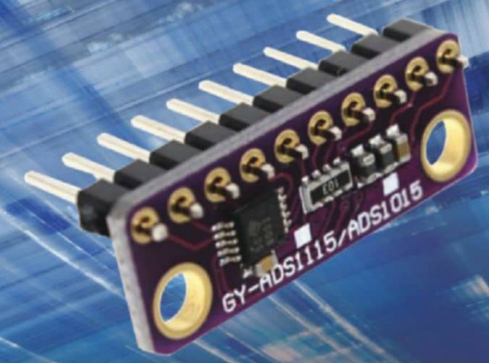


FIGURE 1
This shows one of many similar breakout boards that are available using the ADS1115 ADC. These breakout boards are generally less expensive than buying the ADS1115 chip itself (in small quantities).

While it is somewhat of a moving target, to be considered a high-resolution analog to digital converter (ADC), a device generally had to have a resolution of 16 bits or greater. Currently, many ADCs are available with 16-bit resolution, and 20+ bits could now be considered the norm for the high-resolution designation. Of course, the higher the sample rate of an ADC, the harder it is to achieve high-resolution. In this article, I will be describing ADCs with sample rates in the hundreds to thousands of samples per second, as that is what I generally need in my projects. In this speed range, it is quite common for such ADCs to include a Programmable Gain Amplifier (PGA) and maybe a multiplexer to expand the number of channels that can be measured. When such features are built in, the device may be called an analog front end (AFE) instead of an ADC, so if you are doing a search for high-resolution ADCs on your distributer's website, you may want to search on this term as well.

THE CONTENDERS

Here are the three ADCs that I will be discussing in this article:

- ADS1115 16-bit ADC with PGA and 4-channel multiplexer (Texas Instruments)
- NAU7802 24-bit ADC with PGA and 2-channel multiplexer (Nuvoton)
- MCP3910/3911 24-bit dual ADC with dual PGA (Microchip)

If you look at the number of driver libraries available for each of these devices, written for different MCU platforms, it's clear that the ADS1115 is the most commonly used ADC of the three, at least in terms of the number of different projects it appears in. The NAU7802 is specifically designed for measuring strain gauges, so is a somewhat more specialized device, but highly useful for many signal measurements in the millivolt or microvolt range— particularly if a stable excitation power source is required, since this device contains such a source. The MCP3910/3911 is the only one that contains two discrete PGA/ADCs instead of the single ADC preceded by a multiplexer, as used by the other two devices. It's designed to take two synchronous readings and is specifically targeted at mains power meters. However, it can also be used to measure various AC or DC signals with resolutions down in the microvolt level. Aimed at measuring 50Hz/60Hz AC voltages and currents, with the extremely high accuracy required for utility power meters, it features the ability to sample at much higher sample rates than the other two devices being discussed in this column.

In terms of cost, the NAU7802 is the least expensive, at about $2 USD and the ADS1115 is the most expensive at $5 USD. However, the ADS1115 is so ubiquitous that you can find many different tiny breakout boards containing this device, for less than $5 USD. I use them, as they take up very little PCB space (see **Figure 1**) and eliminate the need for me to solder to a tiny 10 pin VSSOP package. The NAU7802 is the only device of the three that is available in a DIP package, so I like using those as well.

The MCP3910 comes in a 20-pin SSOP package. I discovered this device on a MIKROE-3150 Click board. **Figure 2** shows this Click board— it is also quite easy to work with.

Before looking at these three devices in detail, please refer to **Table 1**. This is an overview of some of the most important features and specifications of the three devices. Notice that both the ADS1115 and NAU7802 use the I²C serial interface. The MCP3910/3911 use an SPI interface but can also be configured to use a two-wire serial interface, which lends itself to isolating the device galvanically with the addition of a two-channel isolator. This is useful when the MCP3910 is being used to measure high AC voltages, as found on mains supplies. The MCP3910/3911 can also provide the ADC modulator data streams for each channel, in case you want to convert these data streams to digital values, using algorithms on your host MCU. The MCP3910/3911 is the only device that has a maximum VDD of 3.6V. The other two can be powered by up to 5.5V and can easily interface to either 5V or 3.3V MCUs.

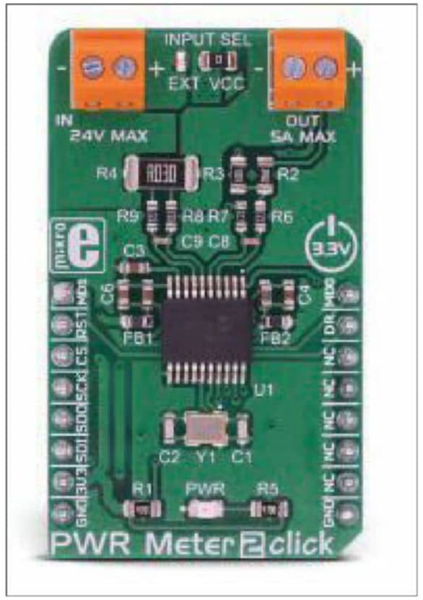


FIGURE 2
This is a MIKROE Click board containing a Microchip MCP3910 dual ADC (alternately called an AFE). While the MCP3910 was designed to measure AC voltages/currents in a mains power monitor, this Click board is configured to measure the same for a low-voltage DC power supply.

Feature/ Specification	ADS1115	NAU7802	MCP3910/3911
Resolution	16-bits	24-bits	24-bits
Power supply	2.0-5.5V	2.7-5.5V	2.7-3.6V
Analog input Channels	4 S.E. 4 Differential	2 Differential	2 Differential
# of ADCs	1	1	2
Internal Vref	Yes	Yes	Yes
Digital Output interface	I²C	I²C	SPI or 2-wire Serial
Sensor Excitation supply	No	Yes	no
Window comparator/ alarm	Yes	No	No
PGA Gains or FS ranges	6	8	6
Gain/Offset calibration	No	Gain	Both
Internal clock	Yes- RC	Yes-RC	Yes-crystal needed
Shutdown mode	No, but runs <150uA	Yes <1uA	Yes <10uA
Sample Rates`	8-860 sps	10-320 sps	35- 125K depends upon crystal freq.
Breakout Board available	Yes- many types	Adafruit/Sparkfun	Mikroelektronika Click
Package	QFN(10), VSSOP(10)	DIP-16, SOP-16	QFN(20) SSOP(20)

TABLE 1
This table provides a feature matrix for the thee ADCs described in this column. Each device has its own strengths and weaknesses.

FEATURES

INPUT VOLTAGE RANGE

Depending upon the signal that you are measuring, the input voltage range of the ADC you choose can be the over-riding parameter. All three devices contain differential inputs, but their input voltage measurement range varies. In the case of both the NAU7802 and the MCP3910/3911, the two differential inputs can also be used as two single-ended inputs by merely connecting the negative inputs to the ADC ground bus. The ADS1115 is somewhat more versatile, containing four inputs which can be configured as either four single-ended, two differential channels or a combination of both.

Let's look at the ADS1115 single-ended case first. In this configuration, any of the 4 inputs will be measured with respect to the ADC's ground bus. The ADS1115 uses a single power supply from 2.0V to 5.5V, and the PGA contained in the device is powered by this 2.0V to 5.5V power supply. The ADS1115's single-ended input voltage range extends down to the ADC's GND bus and up to whatever voltage is being used to power the device (2.0V-5.5V). You cannot apply a negative voltage, with respect to the ADC's GND, and expect it to be measured correctly—it will just return a value of zero. It can also damage the device if it exceeds 0.3V below GND potential. The same can be said if the input signal exceeds the VDD voltage. It will read the full-scale value and may damage the device if it goes beyond VDD+ 0.3V. However, within the GND to VDD range, it will measure the voltage and provide a linear reading— assuming a PGA gain setting of 1x. As the PGA gain increases, the voltage input range will decrease accordingly, but it will always be referenced to GND potential. Negative voltages will always be unmeasurable and read out as 0.

When the ADS1115 is configured for a differential input, the same considerations apply. Neither of the two differential inputs can have their peak excursions extend beyond GND and VDD. So, when powered by 5V, each input can swing between GND and 5V, so the full-scale voltage range that could be measured with a PGA gain of 1x is ±5V. The ADS1115 uses the ±6.144V range for this case. Note however, that on this range, you will never see a digital value approaching either +6.144V or -6.144V, as this voltage exceeds the maximum input voltage range of the ADC, even when it is powered by 5V. Of course, as you reduce the ADS1115's VDD voltage, this input range decreases proportionately. If the ADS1115 were powered by 3.3V, the ±4.096V range would, for example, never show a full-scale reading, because that voltage would exceed the 3.3V maximum allowable input voltage. **Table 2** shows the six full-scale ranges of the ADS1115 and the corresponding weight of the LSB, in terms of voltage.

Next, let's look at the NAU7802. It also has two differential inputs which can be configured for single-ended signal measurements by simply grounding the negative input. Like the ADS1115, the minimum input voltage (without possible damage) on either the plus or minus inputs of each differential channel, is GND-0.3V and the maximum is VDD+0.3V. However, the NAU7802 differs from the ADS1115 in the following respect:

If the NAU7802 is operated in the single-ended mode, (negative input connected to ground) the voltage range that can be accurately measured does NOT include GND or VDD. At those extremes, the readings are no longer linear- i.e. correct. Nuvoton doesn't make this very clear in their datasheet. They do specify the following two cases:

With PGA gain= 64,128
common mode range is GND + 1.5V, AVDD-1.5V

With PGA disabled
common mode range is GND- 0.1V, AVDD+ 0.1V

That leaves the acceptable common mode range for PGA gains between 1 and 32 unspecified. It appears that one could measure voltages down to GND potential accurately with the PGA bypassed. I've not tried this because I generally need to use the PGA for gain purposes. When using the PGA at gains below 64, I still found that readings were not accurate as the input voltage to be measured approached GND. Since that was a deal-breaker, I didn't try measuring voltages that approached VDD, but I suspect they would not be accurate either.

This differs from the ADS1115, as outlined above. I'm certain that the reason for this is that the NAU7802 was designed primarily for measuring strain gauge bridges used in weighing devices. As such, the common mode voltage output of the strain gauge bridge is the excitation voltage/2. The strain gauges themselves produce only millivolt level differential signals when configured in this bridge configuration. So, in this case, the common mode voltage, seen by either the plus or minus inputs to the NAU7802, will never be too far away from the excitation voltage/2: i.e. somewhere between1.2V and 2.25V. Therefore, neither of the NAU7802's differential inputs will ever approach either the GND or VDD potential, where the measurement accuracy is very poor.

The takeaway from this is that the NAU7802 is not the device to choose if you are measuring single-ended signals that go down to zero volts with respect to GND.

The MCP3910 also contains two differential input channels but each one of them has its own ADC. For both its plus and minus inputs, the maximum voltage range is specified as ±1.0V. However, to ensure maximum accuracy, Microchip strongly recommends that you keep this voltage within the ±0.6V range. The important things to note here are:

- This device CAN accurately measure bipolar voltages with respect to GND, in a single-ended configuration.
- It follows that it can accurately measure voltages down to zero, in a single-ended configuration.
- The ±0.6V full scale range can easily be extended to higher voltages by using a resistive divider.

Of the three ADCs covered in this column, only the MCP3910 can handle the scenario shown in the first bullet point, and only it and the ADS1115 can handle the situation described in the second bullet point. This characteristic alone may be what determines which of these three devices you choose for a specific application.

In the next sections, I'll discuss specifics of each of the three devices.

THE ADS1115 (TEXAS INSTRUMENTS)

Seeing the large number of breakout boards available for this device, as well as the large number of driver libraries available for various MCUs, this device appears to be extremely popular. I have come across driver libraries written by the following:

- Adafruit libraries for both the Arduino IDE and their own Circuit Python platform.
- At least seven other independent Arduino libraries, including those written for specific boards made by DFRobot and M5Stack- all available using the Arduino library manager.
- OS-level libraries (Device Tree files) for Raspberry Pi and the BeagleY-AI SBCs.
- Python libraries for Raspberry Pi running Linux.

I've used both the Adafruit Arduino and Circuit Python libraries and they work well, although they don't handle all the features of the ADS1115 (which I'll get into later) Note that the Adafruit library is named Adafruit_ADS1x15- it can also be used with the 12-bit ADS1015, but you must be careful as there are some differences between the two.

I've also tried out the OS-level Device Tree library on both the Raspberry Pi and BeagleboardY-AI. While these work, the whole concept of these Linux drivers— accessing the device as if it were a file, is somewhat foreign to me, and I have my doubts that you can access the ADS1115 at its maximum sample rate of 860 SPS with this kernel level driver. But I stand to be corrected on this point.

The ADS1115 is an I²C device with a base address of 0x48. This detail is taken care of by the driver library itself. However, the ADS1115 also contains an ADDR pin, which allows you to change its address to the next three higher addresses, depending upon how this pin is wired up. See **Table 3** for details. Note that unless you wire this pin to GND, you may have to adjust the base address used by whatever library you choose.

The ADS1115 has eight different sample rates between eight SPS and 860 SPS, with better accuracy at the lower rates. If you are running the ADS1115 at its highest sample rate, the speed of the I²C bus may become a limiting factor. By default, the ADS1115 works fine with the standard I²C clock speeds of 100khz or 400khz. You must initially communicate with the device at either of these two speeds, but you can later send a command to place the device in high-speed mode, after which you can increase the I²C speed up to 1MHz, assuming that your MCU can also handle this higher rate.

One consideration that can easily be overlooked is the correlation between an ADC's sample rate and how many of its channels you need to use. The ADS1115 contains only 1 ADC which is shared by its four channels using an input multiplexer. Logically, you would assume that the effective sample rate would have to be divided by the number of channels being actively measured. In the ADS1115 device , this correlation holds true, and T.I. calls this

Full Scale Range	LSB Size	Differential Input Impedance
±6.144V	187.5µV	22MΩ
±4.096V	125µV	15MΩ
±2.048V	62.5µV	4.9MΩ
±1.024V	31.25µV	2.4MΩ
±0.512V	15.625µV	710kΩ
±0.256V	7.8125µV	710kΩ

TABLE 2
This table shows the various full-scale ranges available on the ADS1115 ADC, as well as the differential input impedance seen by the source, for each range.

ADDR PIN	I²C Address
GND	0x48
VDD	0x49
SDA	0x4A
SCL	0x4B

TABLE 3
This table shows the four different I²C addresses available on the ADS1115, as selected by the configuration of the ADDR pin.

"single cycle settling" in the datasheet. This is not the case for the NAU7802, which I'll discuss in more detail later in the column.

Referring to Table 2, you can see that apart from the 6.144V range, all the other ranges are related by powers of 2. I have used this device in projects where I used auto-ranging and can verify that all the ranges between ±0.256 and ±4.096 are exactly related by a factor of 2.00. So, in this scenario, you can merely adjust the value by exactly 2.00 when you change ranges. No separate calibration constants are needed for each range.

From Table 2, you can also see that for all the upper voltage ranges: the differential input impedance is 2.4MΩ up to 22MΩ. For most Wheatstone bridge applications, these input impedances are unlikely to affect the accuracy of the readings, but on the ±0.256V and ±0.512V ranges, you may have to factor in the lower 710kΩ input impedance into your calculations.

An ADS1115 feature that I have made use of in several projects is its built-in programable comparator. This comparator can be configured in the following modes:

- Triggered when the ADC voltage is greater than some specified value
- Triggered when the ADC voltage is less than some specified value
- Triggered when the ADC voltage is outside a specified voltage window

The resulting trigger signal can be either active-low or active-high (great when triggering an interrupt pin which has a fixed edge configuration). It can also be programmed to be latching or non-latching. Also, you can specify whether 1, 2 or 4 samples must meet the trigger criteria, before this trigger is output to the ALERT/RDY pin. The only shortcoming here is that the ALERT/RDY pin is also used for the Data Ready indication (i.e., conversion complete), so you must pick which of the two signals you need the most. If you choose to use the comparator feature, you can, however, monitor the Data Ready condition by reading an ADS1115 register.

There are four main registers (all 16-bit):

- Conversion Register (ADC conversion result)
- Config Register
- Comparator Low Threshold Register
- Comparator High Threshold Register

The complete configuration of the ADS1115 is handled by setting various bits in the 16-bit Config Register. Therefore, your program code should keep track of the current values of each of the configuration bits, so that when you change one aspect of the ADS1115's configuration, you don't clobber other aspects of the configuration, by disturbing these other bits.

These four registers are accessed by loading the Pointer Register first, which points to the register you wish to access. As the ADS1115 can measure bipolar values, the ADC result is expressed as a 2's complement 16-bit value.

The ADS1115 is an extremely low-power device. It only consumes 150uA while it is operating and can be shut down when not actually measuring, decreasing the current draw even further. Since its operating current is so low, you could power this device using an MCU's GPIO pin, and reduce its power consumption to zero, when not using it for measurements.

THE NAU7802 (NUVOTON)

As mentioned earlier, this ADC was specifically designed to measure strain gauges used in a weighing apparatus (possibly consumer weight scales). These strain gauges are always configured as a Wheatstone bridge and produce differential signals with a large common mode voltage (several volts) and a tiny signal voltage (millivolts down to microvolts, depending upon how much weight is being measured).

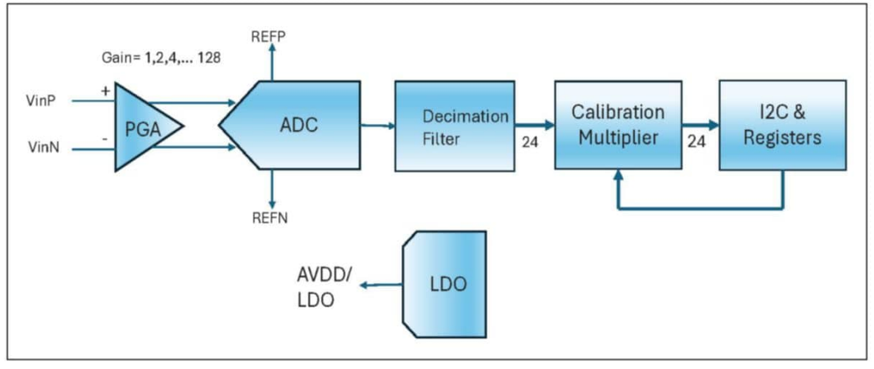


FIGURE 3
This is the block diagram of the NAU7802 ADC. It is the only one of the three ADCs covered here that includes a programmable LDO which provides both a reference voltage for the ADC and a bridge excitation source.

Figure 3 is a block diagram of the NAU7802 (less the input multiplexer). Like the ADS1115, this device uses an I²C bus to communicate with its host MCU. This I²C bus operates at the standard 100kHz clock rate by default. In this default setting, it has internal I²C pull-up resistors of 50k. For higher I²C speeds you must adjust the SPE and WPD bits in register 0x11 (I C CONTROL) to enable 1.6k pull-ups needed for the higher speeds (or add similar value external resistors). The datasheet doesn't specify the maximum I²C clock rate, and I have only used it up to 400kHz.

The NAU7802 (**Figure 4**) contains a PGA with differential inputs, with gains between 1 and 128, in steps of powers of 2. While I have not tested this parameter as thoroughly as I have on the ADS1115, I suspect that the various PGA gain steps are all exactly 2.00 from one gain range to the next higher range. This would eliminate the need for the host MCU to store calibration coefficients for each range. The 24-bit sigma-delta ADC output goes to a decimation filter which is not described in any detail in the datasheet.

The Decimation filter's output can be calibrated in terms of both offset and gain. The Offset Register's value (default is zero) is added to the decimation filter's output value. Similarly, a 24-bit Gain Calibration Register's value stores a gain calibration value (default is 1.0). The decimation filter's output value is multiplied by this value.

There is a calibration command which you can run, which will perform a self-calibration on the NAU7802. This compensates for offset and gain variations in the internal PGA itself- not any external circuitry/sensor that you may have hooked up to it. However, you could also perform a full-circuit calibration using external measuring equipment and/or an accurate simulator for the sensor being used, and calculate the required values needed to be loaded into the offset and gain calibration registers. Note that there are offset and gain calibration registers for each of the two input channels.

The NAU7802 contains a programmable LDO which provides reference voltages in eight different voltages, from 2.4V to 4.5V. This LDO output is used to power the ADC and PGA, and can be connected to the REFP pin, to provide the ADC's reference source. The LDO can supply a maximum of 10mA, so it can also be used as the excitation power supply for the strain gauge, so long as the excitation current requirement is 10mA or less. If you are measuring a bridge circuit, it's advantageous to make the NAU7802's ADC voltage reference the same as the bridge excitation voltage from the AVDD/ LDO terminal. This makes all measurements ratiometric, and the accuracy of the excitation voltage/ADC reference (at the AVDD/LDO pin) is not critical, as it gets cancelled out in the conversion formula.

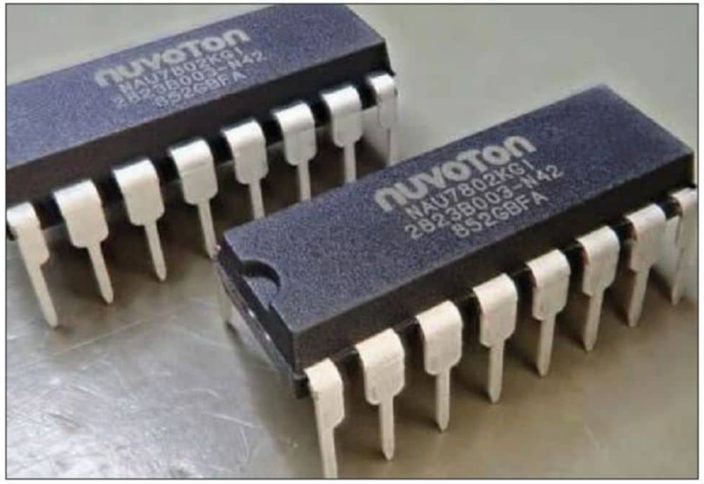


FIGURE 4
The NAU7802 is available in both SOIC and DIP packages-- I like the latter as they are easy to mount

The NAU7802's internal bandgap reference (used by the LDO) has its temperature coefficient stated in the datasheet. The NAU7802 contains a temperature sensor (a silicon diode arrangement) which can be measured by the ADC, when the input multiplexer is switched to this source. You could use this temperature reading to compensate for the temperature coefficient of the bandgap reference, if necessary. However, if you measure a strain gauge bridge, using the ratiometric method mentioned above, this is unnecessary.

Used in a weigh scale, the NAU7802 would likely only need one channel. The NAU7802 contains two differential input channels, selected by a multiplexer. If you are only using one of these channels, the NAU7802 allows you to connect a filter capacitor across the positive/ negative inputs to Channel2. If bit seven of the POWER_CONTROL REGISTER (0x1C) is set, it will place this capacitor across the output of the PGA, to better filter out noise from the signal. I believe that this bit is set by default, which will screw things up if you are using both NAU7802's channels and forget to clear this bit!

Generally, when you are weighing something, its weight will not change rapidly over time, so the NAU7802 doesn't need to have a fast sample rate. In that respect, it's like the ADS1115, having 5 sample rates between 10 SPS and 320 SPS. The NAU7802's clock signal can either be the internal 4.9152MHz RC clock or an external 4.9152MHz crystal. The internal RC clock is accurate within 1%, which allows the NAU7802 to achieve >100dB notch rejection at both 50Hz and 60Hz. This is

FEATURES

important when measuring strain gauges due to their very low output voltage level, as well as the possibility that they may be in a noisy electrical signal environment.

As I touched upon earlier, the sample rate of an ADC containing a multiplexer to handle multiple channels, will be inversely proportional to the number of channels being actively measured. This is true for the ADS1115, which has Single Cycle Settling. That is, the ADS1115 allows you to switch its channel multiplexor and immediately collect and trust the next ADC reading that you get. The NAU7802 does not contain this feature, nor does its datasheet mention this. After quite a bit of experimenting, I determined the following:

- If you wish to use both NAU7802's input channels, all five sample rates must be divided by five. So, if you configure it for a 320 SPS rate, and then switch between the two channels to obtain readings for both, the actual data rate that you will achieve is only 320/5 or 64 SPS.
- You must take a "dummy" ADC reading after switching channels to empty the pipeline of the reading from the other channel.

I discovered this much slower sample rate by measuring the time between the ADC "read" command, and the conversion complete status from the Data Ready pin. This threw me for a loop— although I was aware that this limitation is inherent in the design of some ADCs, I expected this to be noted in the NAU7802's datasheet, which it wasn't.

Figure 5 shows how the NAU7802 would be used with a strain gauge bridge.

I have used the NAU7802 and found Arduino libraries for them as follows:

- Sparkfun- SparkFun_Qwiic_Scale_NAU7802_Arduino_Library.h
- Adafruit- Adafruit NAU7802 Library

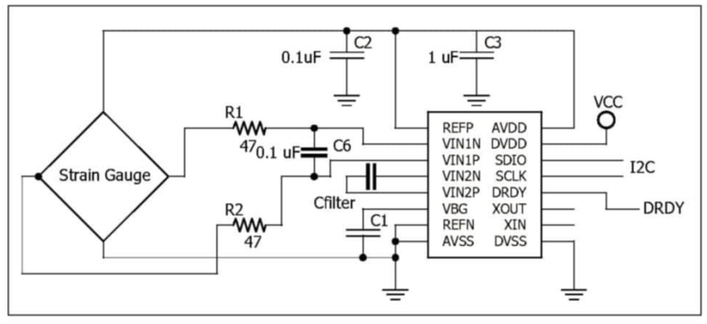


FIGURE 5
This shows how a NAU7802 can be configured to measure a strain gauge bridge. Notice that the ADC's reference voltage as well as the bridge excitation are provided by the AVDD pin (LDO output).

Both are available using the Arduino Library Manager. I have tested the Sparkfun library and found it to work well.

MCP3910 (MICROCHIP)

I just came across this ADC recently when checking out various Click boards made by Mikroe, specifically the MIKROE-3150. There are literally a thousand different Click boards, and there are a lot of sensors and chips available mounted on these boards. I've noticed that some MCU development boards, by various semiconductor companies, come with one or more Click footprints on them, for easy prototyping.

The MCP3910 ADC (**Figure 6**) is made by Microchip and is targeted for use in mains power meters. I later discovered that Microchip also makes the MCP3911, which, on the surface, seems to be almost identical to the MCP3910 in terms of functionality and specifications. However, when I looked over the datasheet closely, I found that the register addresses and bit locations were quite different between the two devices. I'll cover this in more detail when I discuss driver libraries.

The MCP3910/3911 are specifically targeted for use in mains power meters. This is a demanding use case since accuracy is critical—assuming you care what your electricity bill is each month! Unlike the other two ADCs discussed in this article, the MCP3910 is designed to measure AC voltages and currents. To do this properly, i.e. handling Power Factor considerations, requires two discrete PGAs and ADCs instead of using an input multiplexer like the ADS1115 and NAU7802 use. Both MCP3910 Channel1 and Channel two are differential inputs. The device contains an accurate bandgap reference but can be switched to an external reference by setting bit 7 in the CONFIG1 register.

The MCP3910/3911 also feature much higher sample rates than the other two ADCs discussed. This is needed to accurately measure the RMS values of both the mains voltage and current. They feature sample rates up to 125ksps, and their accuracy improves as the sample rate goes down, due to higher oversampling ratios. Unlike the other two ADCs discussed, the MCP3910/3911 requires an external crystal to operate. This can go as high as 16MHz. The MIKROE-3150 schematic shows this Click board containing a 20MHz crystal. When the sample rates that I measured didn't agree with the values I calculated from the datasheet formula, I took a magnifying glass to the crystal and saw that it was actually 12MHz. With this crystal, the maximum sample rate would be reduced to 93.75ksps. Because of the higher sample rates at which this device can operate, it uses an SPI port to communicate with the MCU, rather than the I²C bus used on the ADS1115 and NAU7802.

The PGAs contained in the MCP3910 have gains between 1 and 32 in steps of powers of 2. While I haven't checked this out as I did with the ADS1115, I am assuming that each PGA gain is exactly 2.00 times that of the next lower gain step, eliminating the need for the host MCU to store calibration coefficients for each gain setting. The full-scale values for the different PGA gain settings are ±0.6V with a PGA gain=1, down to ±0.01875V with a PGA gain=32. Reiterating what I mentioned in an earlier section, this is the only ADC of the three covered in this column, that can handle bipolar signals directly. This is true even if the inputs are used in a single-ended configuration (i.e. the negative input is connected to the ADC's GND bus).

The MCP3910 does not contain any circuitry to allow for a self-calibration routine to be performed, as is found in the NAU7802. I suspect that in its targeted use as a mains power meter, the calibration of the entire circuit, including the resistor(s) needed to drop the mains voltage to ±0.6V and the current shunt resistor/coil, would need to be performed. This would make a calibration routine of only the internal ADC components inadequate. The device contains two 24-bit registers for offset and gain calibration, but these values must be loaded by the host MCU, after a full-circuit calibration routine has been performed. The range of the gain calibration goes from 0 to 1.99999999x and this multiplication is performed internally in the MCP3910. The value in the 24-bit Offset register is internally added to the ADC's 24-bit output value. When I was writing the driver library for the MCP3910, I tested out the Offset calibration register.

The MCP3910 is designed to measure AC power and must take into consideration that Power Factor calculations are involved in this. One MCP3910 feature that helps in this regard is its Phase Delay compensation. This feature allows the device to compensate for any phase discrepancies between its voltage measurement (Channel1), and its current measurement (Channel2), which would involve some sort of current shunt which might introduce a phase shift. This Phase Delay compensation has a 1µs time resolution and would be more accurate than any calculation the host MCU might perform, using the sample data alone.

While I had no use for it, the MCP3910 allows for a digital control/output mode which differs from its standard SPI bus. This is a 2-wire serial bus with just Clock and Data

FIGURE 6
The Microchip MCP3910 ADC contains two separate ADCs since it is targeted primarily for mains power meters

signals. In its targeted use as a mains power meter, the chances are that the MCP3910 and associated analog circuitry will be connected to the mains lines and will need to be galvanically isolated from the host MCU. This 2-wire serial bus can be implemented using only two high-speed isolators. In this mode, the device configuration is handled not by setting the MCP3910's 24-bit configuration registers, but by the logic levels present on 5 of the device's pins (some of which perform other functions, when the device is operated in the SPI mode). The 24-bit sample data is streamed out through this 2-wire serial port in a specific frame format.

Along with the 2-wire mode described above, the MCP3910's ADCs can output their raw ADC modulator outputs via the MDAT0 and MDAT1 pins. This would require that the host MCU do the decimation/filtering that is normally performed in the MCP3910 itself. On this topic, the MCP3910 contains $SINC^3$ and a $SINC^1$ decimation filter blocks which are configured according to the oversampling ratio selected.

There are a couple of MCP3910 security features which I did not investigate. There is a 16-bit Cyclic Redundancy Check that can optionally be performed and inserted at the end of a data read sequence. There is also a security register called LOCKCRC (0x1F). At power up, bits 16-23 of this register contain a "password" of 0xA5 (a common alternating bit pattern often used for such purposes). If you write any other value to these 8-bits, all further writes to any of the other registers is blocked. This makes unintentional changes to these registers, due to a host MCU program execution problem, impossible. It doesn't take much imagination to think of cases where a "hacker" might want to re-configure the Gain calibration register with a coefficient that is lower than it should be, such that their power bill is slashed! In my MCP3910 test program,

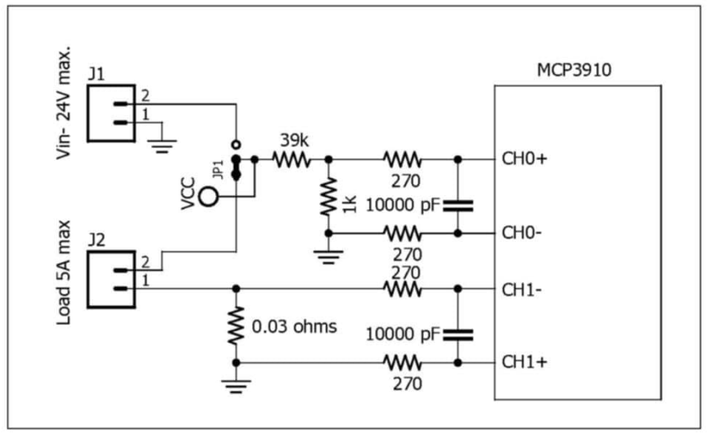


FIGURE 7
This diagram shows the analog circuitry found on the MIKROE-3150 Power Meter 2 Click board, containing a MCP3910 dual ADC.

I read this LOCKCRC register early on and ascertain whether the MCP3910 is present/working or not, depending upon the presence of this passport value.

The work that I did with the MCP3910 was done using a MIKROE-3150 Click board as mentioned earlier. I'll briefly describe the circuitry on this board. The MCP3910's SPI and Data Ready lines are pinned out directly to pins on the Click standard footprint. There is no LDO on the board, so it must be powered by a 3.3V power source, and its logic levels are also at 3.3V levels. It contains a 12 MHz crystal- not the 20MHz crystal shown on the board's schematic diagram.

The MCP3910's Channel1 input is configured to measure a voltage of up to 24VDC via a 2-pin block. It also contains a 2-pin block to which you can attach a load for this supply. The current shunt it uses to measure the load current is a 0.03Ω resistor, and it is placed on the low-side of the load. The voltage across this shunt is applied to Channel2. Given the wattage of this shunt resistor, the maximum safe current is shown as 5A, although that will only correspond to 25% of the MCP3910's full-scale input voltage when using a PGA gain of 1. **Figure 7** is the diagram of the analog portion of this Click board. Note that by default, the board is jumpered to measure its own VCC, and a zero-ohm resistor must be removed and a solder link soldered into place to measure the voltage at the 2-pin block marked IN.

DRIVER LIBRARY

The MCP3910/3911 are complex devices, and I initially wondered whether there were driver libraries for them. Mikroe provides driver libraries for any devices contained on their Click boards that interface to an MCU. These can be found on their website under the heading Libstock. However, Mikroe also sells their own line of compiler toolchains, and these drivers are written for these compilers. Unless you use one of their compilers, you can't use these driver libraries as they are distributed in a file format called mpkg, which I assume is proprietary.

I was only able to find one MCP3911 driver library which was written for the Arduino IDE. Looking closely at both the MCP3910 and MCP3911 datasheets, I found that these two devices use different register definitions- both in terms of register addresses for a given function, as well as the specific bit locations within those registers. Therefore, I've written an Arduino class library for the MCP3910 which is loosely based upon the MCP3911 library written by Ruben Stadler and available on Github (https://github.com/architecture-building-systems/MCP3911_Arduino_Library/blob/master/src/MCP3911.cpp)

I haven't tested the above library as I didn't have a MCP3911 device. However, I've tested the MCP3910 library that I wrote for the MCP3910.

Here is a short description of the functions available in my MCP3910 library.

MCP3910_begin(int CS pin)—This is the first routine you must call and defines the -CS pin used by the SPI port. The defined SPI pins for the target board are used and the SPI is configured for 8MHz clock, Mode 0 and MSB sent first.

MCP3910_Configure(REGISTER SETTINGS settings)—Prior to calling this, you must populate a structure with all the main MCP3910 variables. Then this routine is called with a pointer to this structure. See the MCP3910_Test_BM example program for the register structure needed.

MCP3910_read_chX(uint8_t channel)—Call this routine to retrieve a floating-point number containing the voltage at the channel number specified (PGA gain is factored in).

MCP3910_read_raw_data(uint8_t channel)—Call this routine to get the raw 24-bit 2's complement ADC data for channel specified. To convert this to a voltage reading, you must factor in the full-scale input voltage for the PGA gain that has been configured.

MCP3910_data_to_voltage(long data, uint8_t channel)—This converts the 24-bit raw ADC data into an actual voltage- given the PGA setting configured in the REGISTER SETTING structure.

MCP3910_enter_reset_mode()—This resets the MCP3910 ADCs.

MCP3910_exit_reset_mode()—This takes the MCP3910 ADCs out of the reset state.

MCP3910_write_offset(long offset, uint8_t channel)—This writes a 24-bit offset value to the specified channel's OFFSET register.

MCP3910_write_gain(long gain,uint8_t channel)—This writes a 24-bit value to the specified channel's GAIN register. See the MCP3910 datasheet section 5.13.2 for details on how this 24-bit value is scaled to a multiplier between 0 and 1.9999999x.

MCP3910_write_phase(uint16_t phase)— This writes a 16-bit number to the 12-bit PHASE register. See section 5.9 of the datasheet for details.

MCP3910_read_register(uint8_t register)—This is a general-purpose function to read the specified MCP3910 24-bit register into a long variable. For example, I use it to read the LOCKCRC register (0x1F) to check that its most significant 8 bits are 0xA5, the default Password value at power-up.

This MCP3910 Arduino driver library, as well as an example sketch are available on the Circuit Cellar website support page for this column.

CONCLUSIONS

Each of these three high-resolution ADCs has their advantages and weaknesses. The ADS1115 has four 16-bit input channels, configurable as single-ended, differential or a combination of both. It is inexpensive and readily available on tiny breakout boards. It is the only one of the three devices that contains a digitally programmable comparator, which is very versatile, and can free the host MCU of the need to monitor for fault conditions, for example. Its PGA has six settings providing full scale values between ±0.256V and ±6.144V. It has its own internal RC clock.

The NAU7802 has two channels and a high resolution 24-bit ADC. It contains a built-in self-calibration function. It has an internal temperature sensor which can be read and used to correct for the temperature coefficient of the internal bandgap reference. It contains its own 4.9152MHz RC clock which is accurate enough to allow for very high levels of 50/60HZ noise rejection. It contains a programmable LDO which can provide an excitation supply for strain gauge bridges, for example. It is the least expensive of the three devices considered in this column and is available in a DIP package.

The MCP3910/3911 is probably the most specialized of the three devices, being designed mainly for mains power monitoring. It is the only one of the three to contain two discrete 24-bit ADCs. Its PGAs cover a range of 1x to 32x and its maximum sample rate is much higher than the other two, at 125ksps. Due to this much-higher maximum sample rate, it uses an SPI bus rather than the I²C bus used by the other two. For mains power line monitoring, it can be configured in a 2-wire serial mode, facilitating the use of just 2 isolators for galvanic isolation. It requires an external clock source, which can be just a crystal and 2 capacitors. While the MCP3910 and MCP3911 perform virtually the same function, with similar specifications, the register architecture differs between the two devices, and separate driver libraries are needed.

I hope that you can benefit from my analysis of these three high-resolution ADCs. Since I have now written a driver library for the MCP3910, all three devices have Arduino driver libraries that are readily available.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

RESOURCES

Adafruit | www.adafruit.com

Microchip | www.microchip.com

Mikroe | www.mikroe.com

Nuvoton | www.nuvoton.com

SparkFun | www.sparkfun.com

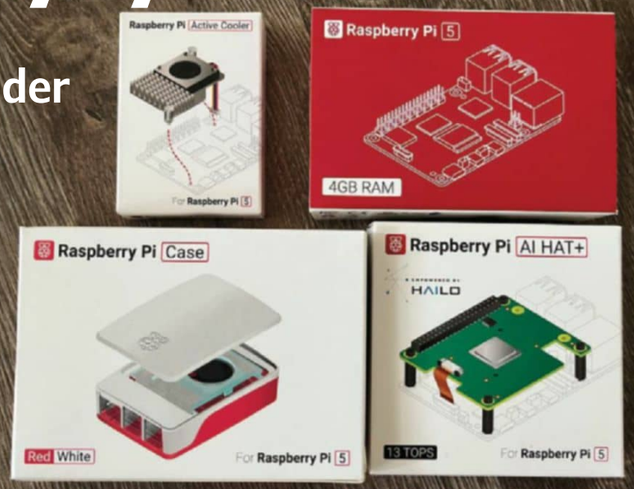
Texas Instruments | www.ti.com

ABOUT THE AUTHOR

Brian Millier runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, NS, Canada) for 30 years.

Create an AI-Driven Smart Home Security System

Part 2—Adding Intruder Detection and Alerts



By
Dhairya Parikh

In Part 1 of this three-part series, Dhairya introduced the new Raspberry Pi AI Kit, set up with the Hailo AI kit hardware and camera module. This month, he begins to build the home security system by adding functionalities for checking the video output of the camera remotely, detecting a person or intruder, and sending an email alert to the user.

Home security is a growing concern, and having a reliable security system can make a significant difference in keeping your home safe. Traditional security systems can be expensive and often require monthly subscriptions, making them inaccessible for many homeowners. However, with the power of the Raspberry Pi 5 and the Hailo AI Kit, you can build an affordable, intelligent security system right at home. This was discussed in Part 1 of this article series (*Circuit Cellar* #416, March, 2025) [1].

The Raspberry Pi 5 is a perfect choice for this project, due to its affordability, compact size, and ability to run AI models efficiently with the Hailo AI Kit. By leveraging AI-powered person detection, you can enhance your security system beyond simple motion detection—ensuring that alerts are only triggered when a person is detected, and reducing false alarms. Setting up the AI kit on our Pi and its capabilities were discussed briefly in Part 1.

In Part 2, I will explain how to build a smart security system by adding person detection and alert functionalities. Once set up, the system will detect intruders and instantly send alerts, allowing you to take immediate action should it be necessary. Additionally, I will explore how to access the camera feed remotely, making it easy to monitor your home from anywhere.

Let's get started with setting up our security system and adding AI-powered intelligence to it.

PROJECT HARDWARE

In this section, I will go over the hardware components required for this project. If you have followed the setup instructions from the Part 1, you should already have needed to continue. Just as a quick refresher, here are the key components:

- Raspberry Pi 5
- Hailo AI Kit for the Pi
- Raspberry Pi Camera or a supported USB Camera

Each component was discussed in detail in last month's article bv. So, at this point, your Raspberry Pi should be set up with the AI Kit and able to run an AI model. If that's the case, you are ready to move forward and start building your home security system. This month, I will focus on adding two essential features that will enhance the system's functionality, and how to implement them.

Person Detection—The system will gain the ability to detect people within the camera frame. Since there are pre-trained AI models available for this task, implementing this feature is straightforward.

Alert Mechanism—Once a person is detected, it's crucial to have a way to notify the user in real time. An alert system will ensure immediate awareness of any potential threats.

With the hardware ready, let's move on to implementing these key features.

BUILDING THE AI-POWERED HOME SECURITY SYSTEM

The first step is to connect all the hardware components together (**Figure 1**), that is, connect the AI kit to the Pi, along with the camera module of your choice. Once this is done, you can start writing the code to add the new features.

But before starting with the actual code development, let's get an idea of how these features will be implemented into the system, with the help of resources and examples.

Person Detection: This feature will not require much setup, since the Raspberry Pi can already run the YOLO models utilizing the power of the Hailo AI kit, which lets it recognize people in a given image frame.

In the example from Part 1 (shown in **Figure 2**), the model was able to recognize people in the frame. We will use the same code as reference, and build on top of it. The example code can be found on GitHub [2].

Alert Mechanism—Setting Up an Email ID for Alerts: For the purposes of this example, I will focus primarily on Email Alerts (though text and push notifications are possible as well). So, email alerts will be sent to a pre-configured email address whenever an intruder is detected by the camera. This will include setting up an Email ID for sending email alerts through the Python code that you write. Let's do that before we start writing code so that we have all the necessary prerequisites. Follow the steps given.

1) Create an email account. I will be using a Gmail account for this tutorial, since it is the easiest to set up for this process. You will need an account with 2-step verification enabled. I recommend creating a dedicated account for this alerts process.
2) Login to your Gmail account.
3) Once you have logged in, from your Account Settings, go to the following window: Security Tab >> 2-Step Verification >> App passwords. This will open a dedicated window, where you can manage your app passwords. (see **Figure 3**).
4) Enter an App name of your choice in the text box and press on the "Create" button to create your App password. This will

FIGURE 1
A look at the assembled hardware with the Pi case. This includes the active cooler, AI kit, and the Pi camera, with everything mounted on the Raspberry Pi 5.

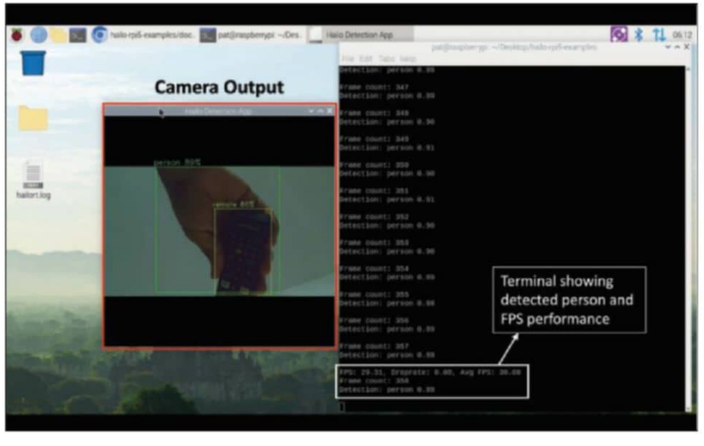


FIGURE 2
Object detection model successfully running on the Raspberry Pi with the AI kit, from the Part 1 of this series [1]. Note that the model is able to detect people, evident from the output on the feed and terminal.

FEATURES

open a new dialog box that will show you your newly created password.

5) Make sure that you store the password before closing the window, because you won't be able to access it again.

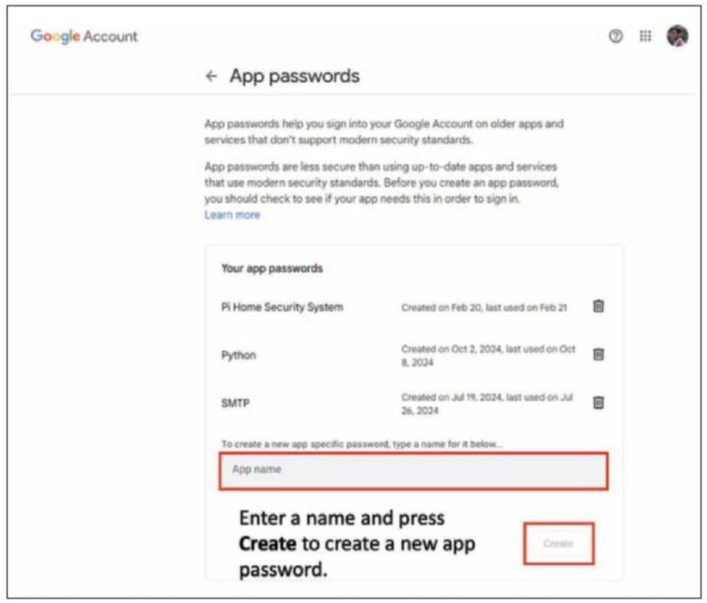

Google Account
App passwords
App passwords help you sign into your Google Account on older apps and services that don't support modern security standards.
App passwords are less secure than using up-to-date apps and services that use modern security standards. Before you create an app password, you should check to see if your app needs this in order to sign in.
Learn more
Your app passwords
Pi Home Security System
Created on Feb 20, last used on Feb 21
Python
Created on Oct 2, 2024, last used on Oct 8, 2024
SMTP
Created on Jul 19, 2024, last used on Jul 26, 2024
To create a new app specific password, type a name for it below...
App name
Enter a name and press Create to create a new app password.
Create

FIGURE 3
The App passwords section of your Gmail account that you will be using to create a password for sending emails from your Python code.

```
import gi
gi.require_version("Gst", "1.0")
from gi.repository import Gst
import cv2
import hailo
import logging
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime
import requests
from hailo_apps_infra.hailo_rpi_common import (
    get_caps_from_pad,
    get_numpy_from_buffer,
    app_callback_class,
)
from hailo_apps_infra.detection_pipeline import
GStreamerDetectionApp

# Configure logging
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)
logger = logging.getLogger(__name__)
```

LISTING 1
Importing all the required libraries and configuring logging.

You now have everything needed to start developing your application code.

CODE WALK-THROUGH

I will walk you through all the code that you need to start your Home Security System with person-detection and alert-mechanism features. Additionally, some changes in the source code will enable it to stream your video feed within your local home network (fancy word for your Wi-Fi connection).

Note that all the code discussed here is available in a dedicated GitHub repository for this article [3]. Three major files must be created for this project:

- `home_security_system_core.py`–Handles the core functionality of your security system, including running the AI model, detecting people, and sending email alerts when a person is detected.
- `stream_security_video.py`–Sets up a local flask server to stream the security feed, allowing anyone on the home network to access it.
- `index.html`–Serves as the web interface, displaying the live security feed from your system.

Also, you need to make a small change in one of the existing Hailo infra files to enable video streaming to your web interface. The first file is "home_security_system_core.py."

HOME SECURITY SYSTEM CORE CODE

As mentioned earlier, this file will be based on the original detection example from Part 1 (provided in the Hailo examples [4]).

First, let's start by importing all the required libraries for this code (**Listing 1**). Note that the following four imports are new, compared to the demo code: 1) "logging "(for logging the relevant info); 2) "smtplib/email "(to configure and section email alerts); 3) "datetime" (to timestamp the detection times); and 4) "requests" (to send detection information to our hosted streaming service). After that, you just configure logging for your code by stating the format in which you want your logs, and then initialize the logger object to use for later.

Second, define a callback class (**Listing 2**), as shown in the demo code to configure email alerts. The code for this is quite long so I am not including the full code here. Please see the GitHub repository for this branch [3] to review the entire code. The important thing to note is that the code involves setting up the following parameters:

```
self.smtp_server = "smtp.gmail.com"
self.smtp_port = 587
self.sender_email = "sender_email@gmail.com"
self.sender_password = "sender_password"
self.receiver_email = "receiver_email@gmail.com"
# Detection parameters
self.last_alert_time = datetime.min
self.alert_cooldown = 300 # 5 minutes between alerts
self.confidence_threshold = 0.6
self.location_name = "Main Entrance"
```

Essentially, you need to configure the following:

- ***Sender and Receiver Email Addresses and the Sender Email Password*** —Created in the previous section.
- ***Alert Cooldown***—Parameter to configure how long to wait before sending out another email (you don't want emails every second, right?). I have set it to 5 minutes.
- ***Confidence Threshold***—Defines the least confidence score our model should have while detecting people, to raise an alert.
- ***Location Name***—A fixed value parameter that can be changed if you are using multiple systems in your home, to distinguish among them.

Once you have set this up, whenever a person is detected in the camera stream, you will get an email alert with the details on detected people. **Figure 4** shows what will the alert will look like.

Third, you need to write your very own "app_callback "function, which is essentially the code that is run for each camera frame (Listing 2). This function does the following:

- Extracts and fetches the frame from the camera feed.
- Converts it into a format that can be understood by your Hailo chip.
- Runs the YoloV5 detection model on this processed frame.
- Based on the results, checks all the objects that have been detected, and draws bounding boxes and adds labels for those objects in the frame.
- Checks if a person/people were detected, and then check if the detection confidence is greater than the user

```
def app_callback(pad, info, user_data):
    buffer = info.get_buffer()
    if buffer is None:
        return Gst.PadProbeReturn.OK

    format, width, height = get_caps_from_pad(pad)
    frame = None
    if user_data.use_frame and format and width and height:
        frame = get_numpy_from_buffer(buffer, format, width, height)

    roi = hailo.get_roi_from_buffer(buffer)
    detections = roi.get_objects_typed(hailo.HAILO_DETECTION)

    detection_info = []
    for detection in detections:
        confidence = detection.get_confidence()
        if detection.get_label() == "person" and confidence > user_data.confidence_threshold:
            bbox = detection.get_bbox()
            track = detection.get_objects_typed(hailo.HAILO_UNIQUE_ID)
            track_id = track[0].get_id() if track else 0

            detection_info.append({
                "id": track_id,
                "confidence": confidence
            })

            if frame is not None:
                x1, y1, x2, y2 = map(int, [bbox.xmin * width, bbox.ymin * height,
                                           bbox.xmax * width, bbox.ymax * height])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, f"ID: {track_id}", (x1, y1-10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    if detection_info:
        user_data.send_alert(detection_info)

    if frame is not None:
        user_data.set_frame(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

    return Gst.PadProbeReturn.OK
```

LISTING 2
App Callback Function: Code that runs for every frame in your video stream.

FEATURES

FIGURE 4
A snapshot of the email that will be sent from your home security system every time an intruder is detected. Note that it can also detect the number of people detected and provide an exact time and date for the last active detection.

SECURITY ALERT: Intruder Detection

Incident Details

Location:	Main Entrance
Date:	February 21, 2025
Time:	18:50:07
Total Persons Detected:	1

Detection Summary

Detection ID	Confidence Score
1	0.93
...	

configured confidence score. If that is the case, compiles a list of all the people detected, giving each a unique ID.
- Once that is done, sends out the email alert with the collected detection results.
- Finally, sends this frame to your display pipeline, which streams it to your server. (More on this later.).
- Finally, returns an "okay" acknowledgement to the GStreamer app, so that it can proceed with the next frame.

Phew! That was a handful, but it's important to understand what is happening in the code, so that if you want to integrate this into your own projects, you don't face any difficulties.

Fourth, all that remains now is initializing the callback class and function, and starting the GStreamer application. This can be done with just 4 lines of code:

```
if __name__ == "__main__":
  user_data = IntruderDetectionCallback()
  app = GStreamerDetectionApp(app_callback, user_data)
  logger.info("Starting Intruder Detection System")
  app.run()
```

This is all for the contents of the main code. You can use the same command as the one given in Part 1 to run this code. Note that you will need to activate the virtual environment we created in last month's article to access the Hailo SDK. Run the following commands to start the application:

```
def DISPLAY_PIPELINE(video_sink='autovideosink', sync='true', show_fps='false', name='hailo_display'):
    """
    Creates a GStreamer pipeline string for displaying the video.
    It includes the hailooverlay plugin to draw bounding boxes and labels on the video.

    Args:
        video_sink (str, optional): The video sink element to use. Defaults to 'autovideosink'.
        sync (str, optional): The sync property for the video sink. Defaults to 'true'.
        show_fps (str, optional): Whether to show the FPS on the video sink. Should be 'true' or 'false'. Defaults to 'false'.
        name (str, optional): The prefix name for the pipeline elements. Defaults to 'hailo_display'.

    Returns:
        str: A string representing the GStreamer pipeline for displaying the video.
    """
    # Construct the display pipeline string
    display_pipeline = (
        f'{OVERLAY_PIPELINE(name=f"{name}_overlay")} ! '
        f'{QUEUE(name=f"{name}_videoconvert_q")} ! '
        f'videoconvert name={name}_videoconvert n-threads=2 qos=false ! '
        f'{QUEUE(name=f"{name}_q")} ! '
        'videoconvert ! openh264enc ! mpegtsmux ! rtpmp2tpay ! udpsink host=127.0.0.1 port=5010'
    )

    return display_pipeline
```

Changing this line changes the target location where the frames are sent.

FIGURE 5
This is the DISPLAY_PIPELINE function from the GStreamer helper functions that you will have to modify to stream your camera output on a local Web interface using UDP sink. The highlighted line is the one that was changed.

```
cd <location-of-examples-repo>/hailo-rpi5-examples
source setup_env.sh       // To start the virtual environment.
python <location-of-article-repo>/home_security_system_core.py --input rpi
```

They will mostly probably not run, since we still need to configure our streaming server. However, how do we send the output feed to our server, instead of showing it on our Pi as we did before?

This was a bit tricky to figure out, because there is no documentation for how to configure sharing your frames to some other source. After hours of research and going through Hailo community posts, I finally figured out a work-around for this.

You will share the frames of your program to a UDP connection (a type of network connection you can access from other sources, like your flask server). For this, you need to change a single line in the following file (see **Figure 5**):

```
hailo-rpi5-examples >> venv_hailo_rpi5_examples >> lib >>python3.11 >> site-packages >> hailo_
apps_infra >> gstreamer_helper_pipelines.py
```

```
from flask import Flask, Response, render_template, request, jsonify
import cv2
from datetime import datetime
import json

app = Flask(__name__)

# Global variable to store the latest detection data
latest_detection = {
  "timestamp": None,
  "location": None,
  "detections": [],
  "total_persons": 0
}

def generate_frames():
  cap = cv2.VideoCapture("udp://127.0.0.1:5010", cv2.CAP_FFMPEG)
  while True:
    success, frame = cap.read()
    if not success:
      print("Unable to read frame.")
      break
    else:
      ret, buffer = cv2.imencode(".jpg", frame)
      frame = buffer.tobytes()
      yield (b"--frame\r\nContent-Type: image/jpeg\r\n\r\n" + frame + b"\r\n")

@app.route("/")
def index():
  return render_template("index.html")

@app.route("/video_feed")
def video_feed():
  return Response(
    generate_frames(), mimetype="multipart/x-mixed-replace; boundary=frame"
  )

@app.route("/update_detection", methods=['POST'])
def update_detection():
  global latest_detection
  data = request.json
  latest_detection = data
  return jsonify({"status": "success"})

@app.route("/get_detection")
def get_detection():
  return jsonify(latest_detection)

if __name__ == "__main__":
  app.run(host="0.0.0.0", port=8080)
```

LISTING 3
Python code for setting up a Flask application that allows you to stream your camera output with detection results on a local network.

FEATURES

FIGURE 6
This is how your Web interface will look after both the streaming and home security code files are running. You will be able to see the detection information as well in real time.

You need to change the line 262 (the last line of the DISPLAY_PIPELINE function) from:

```
'fpsdisplaysink name={name} video-sink={video_sink} sync={sync} text-overlay={show_fps} signal-fps-measurements=true'
```

to:

```
'videoconvert ! openh264enc ! mpegtsmux ! rtpmp2tpay ! udpsink host=127.0.0.1 port=5010'
```

The new line converts the frames to the required format, and sends them over a UDP connection, which enables other applications within the same network to access it. We will now check out the code for the "stream_security_video.py" file, which allows you to create a web interface to access your video stream and also check out the detections there.

STREAM SECURITY VIDEO CODE

This is a very simple code that hosts a flask server (a lightweight web application) where you can view your stream and check out the detection results in near-real time. The code for this is pretty straightforward (**Listing 3**).

As mentioned before, it captures the UDP video stream with OpenCV, and streams it in real time to an endpoint "/video_feed," and this is shown on the "/" endpoint, using an additional file called "index.html." Moreover, there are "get" and "update" detection endpoints as well, which are used by your first program to send the detection results and display them on the dashboard.

The index.html code is vanilla HTML code, so there isn't much to cover. All you need to know is that it creates the interface that you see on the web interface. You can take a look at it in the code repository [3]. Note that it needs to be in the "templates" directory for the flask application to read it—which is where you will find it in the repository as well.

Once you have set this up, just run the following command to start the flask server:

```
python <location-of-article-repo>/stream_security_video.py
```

After starting both the programs, you should be able to view this feed via the web interface. To access this feed, you will have to open the following webpage on any of your devices that are connected to the same network as your Pi. You can get the Pi's IP address by running `hostname -I` on a terminal:

```
http://<raspberry-pi-ip-address>:8080
```

Additionally, you will see the live camera feed, along with detected object bounding boxes and the intruder detection results, on the interface. Refer to **Figure 6** to see how the interface will finally look with a live stream and detection details.

To get a better idea of all the things you log in your home security system code, take a look at the given log snapshot (**Figure 7).**

All the INFO logs are generated by the code you have written. Apart from the camera initialization and configuration, the last two lines (shown in boldface) indicate that we also log when an alter email is sent to us.

```
2025-02-22 13:17:15,354 - INFO - Initialization successful.
2025-02-22 13:17:15,354 - INFO - Camera now open.
2025-02-22 13:17:15,357 - INFO - Camera configuration has been adjusted!
[0:03:46.761679360] [2456]  INFO Camera camera.cpp:1202 configuring streams: (0) 1280x720-RGB888 (1) 1280x720-RGB888 (2) 1920x1080-GBRG_PISP_COMP1
[0:03:46.761841379] [2460]  INFO RPI pisp.cpp:1484 Sensor: /base/axi/pcie@120000/rp1/i2c@80000/ov5647@36 - Selected sensor format: 1920x1080-SGBRG10_1X10 - Selected CFE format: 1920x1080-PC1g
2025-02-22 13:17:15,358 - INFO - Configuration successful!
Picamera2 configuration: width=1280, height=720, format=RGB
2025-02-22 13:17:15,480 - INFO - Camera started
picamera_process started
2025-02-22 13:17:32,310 - INFO - Alert email sent and detection data updated
2025-02-22 13:22:30,728 - INFO - Alert email sent and detection data updated
```

FIGURE 7
Snapshot of log from Home Security System.

CONCLUSION AND NEXT STEPS

That is it for this article! Using your Pi and the AI kit, you have finally built a functional home security system that can be deployed right now! It has intruder detection capabilities from live video feed, the capability to send alerts whenever an intruder is detected, and a Web interface, for your real-time video stream.

However, a few more things can be added to make this system even better. They are face recognition and the ability to detect deliveries. Exciting, right? I will cover those things in the Part 3 of this article series. I'll see you next time!

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] to [4] as marked in the article can be found there.

RESOURCES

Raspberry Pi Foundation | www.raspberrypi.org

Hailo | www.hailo.ai

ABOUT THE AUTHOR

Dhairya Parikh is a seasoned Technical Writer with more than five years of expertise in IoT, Machine Learning, and Hardware. He has authored a book on Raspberry Pi, published numerous technical articles, and currently contributes to RaspberryTips, a leading Raspberry Pi blog. With a talent for simplifying complex topics, Dhairya creates engaging step-by-step guides for developers and tech enthusiasts. He has completed his master's in computer engineering from the University of Waterloo, specializing in AI and Machine Learning.

19843342

TECH FEATURE

Automotive Systems

The Integration of AI and Connectivity in Modern Vehicles

By
Caleb Smith

FIGURE 1
The 1968 Volkswagen Type III Fastback, the beginning of automotive computerization.

The automotive industry is undergoing a profound transformation, evolving from mechanical constructs into intricate software-driven platforms that leverage advanced technologies such as V2X communication and AI. This article examines the latest innovations in automotive automation, clarifying their implications for safety, efficiency, and the future of interconnected mobility in a rapidly advancing technological landscape.

With a history of over 200 years, the automobile has become one of the largest industries in the world, undergoing many changes in its time. It wasn't until 1968, when Volkswagen replaced the carburetors in the type III fastback (**Figure 1**) with an electronic fuel injection system with fuel pressure and speed sensors to control fuel output that automobiles even became computerized. In recent years however, the rate of change in automotive technology has increased at an exponential rate with vehicle connectivity, with AI driven safety and autonomous driving, and a plethora of new sensors allowing cars to take in more information than ever before, cars are no longer merely mechanical tools for transportation, but software driven electronic platforms for innovation to push the bounds of and perfect technological capabilities. The classic American symbol of freedom and individuality is now a symbol of the use of interconnected technology in ways unimaginable two decades ago. In this article, I will highlight some of the newest and most exciting automotive technologies and the driving forces behind them; from V2X communication technologies, Edge AI, autonomous driving, cybersecurity, over-the-air(OTA) updates, smart sensors, and the human-machine interfaces that enable more vehicle control than ever before.

V2X COMMUNICATIONS: THE NEW WAVE OF INTERCONNECTIVITY

Let's begin by taking a look at one of the hottest new buzzwords in automotive technology: V2X communications. Short for Vehicle to Everything, V2X communications are the systems by which a vehicle may connect with its environment, including other vehicles (V2V), pedestrians (V2P), surrounding infrastructure (V2I), and wireless networks (V2N) such as Wi-Fi, Bluetooth, cellular data, and GPS, as seen in **Figure 2**. This is typically achieved through visual sensors, LiDAR, radar, and wireless communications technology such as those developed by STMicroelectronics driven by AI-based edge computing to locally process data to reduce the need for server connection and the latency of decision making. The main benefits of this are obvious: increased pedestrian safety and driver safety, and increased traffic efficiency. Let's take a look at these subsections individually and examine their workings and benefits:

- V2V communication is the communication between equipped vehicles that allows them to sense the speed, position, and direction, sometimes in addition to other data of other vehicles on the road. This enables not just early warnings of potential accidents for drivers but also enables autonomous vehicles, or vehicles operating with driver assistance to respond to others on the road when merging lanes, braking, or turning in an advanced collision avoidance system.
- V2I communication is the communication between a vehicle and the surrounding infrastructure, such as stop lights and road signs. Either through AI-guided image recognition software or through wireless communication, this technology improves traffic conditions by decreasing congestion and providing real-time updates on road conditions, traffic, and speed limits.
- V2P communication is the communication between a vehicle and surrounding pedestrians. Typically achieved through visual sensors feeding into an AI-driven image recognition software trained to identify pedestrians or cyclists, or by detecting mobile devices, this system enhances pedestrian and cyclist safety, a critical feature in populated areas.
- V2N communication is the communication between a vehicle and any wireless network. At the consumer level, this may allow for real-time GPS navigation, wireless communication for the purposes of personal communication, and weather updates. On the industry level, however, this allows for instantaneous delivery of OTA updates from manufacturers, or enhanced fleet management.

FIGURE 2
The interconnectivity of V2X communication.

EDGE COMPUTING AND AI

Edge computing is the process by which computation is moved from the cloud toward the end user, typically on a local device, examples of which are seen in **Figure 3**. Enabled by the more powerful MCUs produced in recent years, such as the NXP i.MX series of processors with AI and machine learning acceleration, and Texas Instruments TDA4VM processor family for real-time AI processing, edge computing seeks to reduce latency, and enhance security by removing cloud computing's reliance on external servers. Enhancing real-time responsiveness, which is crucial for safety-critical applications.

One of the biggest uses of edge computing in automotive systems is in the realm of autonomous driving (which will be discussed in more detail in the next section) where it enables all steps of the automation of driving:

Sensor fusion: Edge computing integrates data from LiDAR, radar, cameras, and ultrasonic sensors to detect obstacles, pedestrians, and traffic signals.

Object detection and recognition: AI models running on embedded systems identify vehicles, road signs, and lane markings.

Decision making algorithms: AI-powered Electronic Control Units (ECUs) predict and respond to traffic conditions in real time, based on the information collected from sensors and analyzed locally, enabling the vehicle to take action without the need for human input.

AI algorithms are making significant strides in enhancing vehicle performance and safety. For instance, Convolutional Neural Networks (CNNs) are widely used for image recognition tasks, enabling vehicles to identify

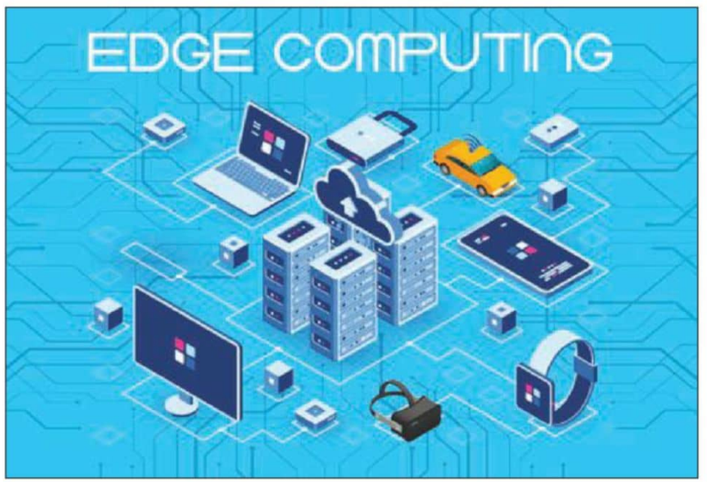


FIGURE 3
Edge computing reduces the need for the cloud and brings computing closer to the end user.

FIGURE 4
A Zoox robotaxi exemplifies the use of autonomous vehicles as a means of public transportation, without the need for a human driver.

road signs, pedestrians, and other critical objects in their environment. Additionally, Reinforcement Learning (RL) algorithms are employed in autonomous driving systems, allowing vehicles to learn optimal driving strategies through a trial-and-error process in simulated environments. These algorithms not only improve the accuracy of object detection but also enhance decision-making processes in complex driving scenarios, contributing to safer and more efficient driving experiences.

In recent years, one feature that has been made possible though edge computing and the implementation of AI has been the implementation of smart diagnostics in commercial and consumer vehicles. By taking in data from sensors within the engine and relevant systems of the car, a local MCU such as Inineon's AI-powered microcontrollers can analyze components, and predict potential failures in critical systems, such as the brakes, battery, or fuel system. This significantly reduces the strain on the vehicle and seeks to improve the service life of the car, while simultaneously improving driver and pedestrian safety by reducing the number and severity of accidents caused by equipment failure.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

Resources

Infneon | www.infineon.com

NVIDIA | www.nvidia.com

NXP Semiconductor | www.nxp.com

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

Tordex | www.tordex.com

VersaLogic | www.versalogic.com

AUTONOMOUS DRIVING

One of the most touted leaps forward in automotive technology of the past decade has been the "self driving car." While a seemingly simple concept, there is disagreement on the definition of "self driving." To remedy this, the Society of Automotive Engineers (SAE) has defined six levels of automotive driving automation:

- Level 0 (No Automation): Human driver handles all tasks
- Level 1 (Driver Assistance): Some driver support from features such as adaptive cruise control
- Level 2 (Partial Automation): Vehicles assist with steering and acceleration but human supervision is required, such as with the Tesla Autopilot feature
- Level 3 (Conditional Automation): Vehicles can perform most driving tasks but require human intervention in certain situations, such as with Honda's Level 3 system, while Level 4 and 5 vehicles do exist, most commercial vehicles designed to be self-driving only operate as high as level 3
- Level 4 (High Automation): Fully autonomous in predefined conditions such self-driving taxis in specific geofenced areas, such as the Zoox robotaxi (**Figure 4**)
- Level 5 (Full Automation): No human intervention required in any scenario

The driving forces behind all self-driving cars is sensor input and information processing. Through embedded sensors such LiDAR, radar, GPS, and V2X communications technologies, autonomous vehicles sense their environment and send the data they collect to an onboard processor, such as NVIDIA's DRIVE Orin system on a chip capable of performing 354 tera operations per second (TOPS), for AI-driven edge computing to determine the best course of action for their situation whether by detecting and classifying objects, planning paths, or controlling vehicle systems such as acceleration and braking. With these features, a significant reduction in automotive collisions may be expected in the coming years, significantly improving public safety, and reducing individual consumer risk.

In this article, I have mentioned many different types of sensors. The reason for this is that in order to enable any process in a self-driving car related to the actual driving of the car in the world requires the car to be able to sense the world around it. For this reason, a wide array of different smart sensors are required for a functional self-driving car. These smart sensors not only enable self-driving cars, but also improve human operated cars by increasing safety, efficiency, and performance by enabling a more precise awareness of the environment and the functionality of the car.

Some of the more common types of smart sensors in modern cars include:

LiDAR: short for Light Detection And Ranging, this system sends out light (typically infrared) and measures the time it takes to return to determine the range of an object. If multiple beams are sent out and measured, it can form a 3d map from the scan, allowing for fast and accurate detection of surrounding objects.

Infrared sensors: By emitting and detecting invisible infrared light, these cameras can serve as an active night vision for the car, without emitting visible light that may distract other drivers.

Inertial measurement units: IMUs use accelerometers such as the ADXL1002 seen in **Figure 5** and gyroscopes to determine the forces acting upon the body of the vehicle at different points. Using basic kinematics, these can provide accurate and reliable data on movement and stability of the vehicle, necessary for computerized decision making.

While autonomous driving may be an obvious use for these sensors, many other practical applications do exist. Even in human operated cars, these sensors aid in collision avoidance by detecting probable collisions and automatically enabling braking systems faster than human reaction time would allow. This system, already in place for some time now, has greatly reduced accidents among the cars in which it is equipped. For engine performance, the inclusion of air quality sensors such as barometers and O_2 sensors can detect the pressure and concentration of Oxygen in the air and adjust fuel injection rate to match it for a stoichiometrically complete burn, increasing engine performance and improving fuel economy. With internal cameras and AI-driven edge computing software, driver monitoring is now also a possibility. By monitoring the driver, alerts can be given to avoid distracted driving: for instance, a warning message may be given if the camera detects that the driver is using a cellphone. Additionally, this same system may issue an alert if it detects that the driver has become tired or fatigued. Systems like this are already in use for fleet vehicles such as buses, taxis, and ambulances.

In this article, we have seen some of the latest advancements in automotive automation. From the automation of driving to the new sensors and communications technology that enable cars to be safer and more efficient than ever before, automotive technology has come a long way in recent years. Cars are no longer mechanical constructs, but multifaceted monuments of engineering and software displaying some of the best technologies of our day. From V2X communication, AI-driven edge computing, autonomous driving, and the ever increasing array of advanced sensors, the complexity of consumer of industry vehicles grows with each passing model. The ability of a vehicle to sense many aspects of its environment, even those undetectable to its driver, has made cars more safe and reliable than ever before. V2X communication has given cars the ability to communicate and interact with other vehicles, infrastructure, pedestrians, and networks, enhancing the efficiency of traffic flow and reducing motor vehicle collisions. AI-driven edge computing has turned cars into self-contained AI-powered decision making machines, previously thought to be impossible in past decades. All of these advancements together, have made autonomous vehicles a reality for more and more people, improving efficiency, and reducing the workload of human drivers, and providing the milliseconds of improved reaction time, unceasing focus, and perfect navigation systems required to prevent accidents, and ensure a safe and comfortable journey for the rider.

With all the recent advancements in AI technology, and the improvement of embedded systems such as the sensors relied on in these cars, I cannot wait to see what the future holds for this symbol of innovation, perseverance, and exploration. Safer, more efficient cars and improved driver experience are surely the wave of the future, reshaping our ideas of safety, mobility, and technology, and making us more connected.

FIGURE 5
ADXL1002 smart chip accelerometer, one of the many smart sensors necessary in modern cars.

TECH FEATURE

Datasheet: 32-Bit Micro controllers

Performance, Integration, and Power Efficiency

By
Caleb Smith,
Editor-in-Chief

DATASHEET

The landscape of 32-bit microcontrollers is undergoing a dramatic shift, driven by the pursuit of performance, integration, and power efficiency. Pushing the boundaries of what's possible in embedded systems, creating new possibilities for innovation.

The landscape of embedded systems is undergoing a dramatic transformation, driven by the evolution of 32-bit microcontrollers (MCUs). These tiny yet powerful devices are no longer just simple control units; they are becoming sophisticated computational platforms capable of tackling increasingly complex tasks with unprecedented efficiency. This evolution is fueled by three key trends: a relentless pursuit of performance, an explosion in integration, and a laser focus on power efficiency.

Today's 32-bit MCUs boast processing power that rivals desktop computers of a decade ago. This is achieved through advancements in processor architectures like ARM Cortex-M series and RISC-V, which offer high clock speeds, optimized instruction sets, and powerful data processing capabilities. These advancements allow for real-time control of complex systems, sophisticated signal processing algorithms, and even advanced machine learning applications at the edge.

The concept of "system-on-a-chip" (SoC) has revolutionized 32-bit MCU design. Modern MCUs integrate a plethora of peripherals and functionalities onto a single chip, eliminating the need for external components and simplifying design complexity. This includes high-speed analog-to-digital converters (ADCs), digital-to-analog converters (DACs), advanced communication interfaces (CAN, SPI, I²C, UART), and even dedicated hardware accelerators for specific tasks like cryptography, motor control, and image processing. This level of integration significantly reduces board space, component count, and power consumption, making it ideal for space-constrained and energy-sensitive applications.

The demand for battery-powered and energy-harvesting applications has driven the development of ultra-low-power (ULP) MCUs. These devices consume minimal power even when performing complex tasks, enabling them to operate for years on a single battery or harvest energy from ambient sources. Achieved through a combination of innovative design, such as dynamic voltage scaling, sleep modes, and power-gating, which manage power consumption based on the workload.

These advancements are driving a wave of innovation. 32-bit MCUs are the backbone of the Internet of Things (IoT) revolution. Their low power consumption and advanced communication capabilities make them ideal for sensor networks, smart home devices, and wearable technology. From advanced driver-assistance systems (ADAS) to electric vehicle control units, they are transforming the automotive landscape. Their robust performance, real-time capabilities, and safety features make them essential for the development of autonomous vehicles. 32-bit MCUs are driving the automation of industrial processes, from robotics and machine control to process monitoring and predictive maintenance. Their high-performance capabilities and integration of advanced communication protocols enable the development of intelligent and efficient industrial systems. Medical devices, including wearable health monitors, implantable devices, and advanced diagnostic equipment are also showing a surge in the incorporation of 32-bit MCUs in their development of. Low power consumption, accuracy, and safety features make them ideal for these critical applications.

The evolution of 32-bit MCUs is far from over. Expect to see continued advancements in processor architectures, integration levels, and power efficiency, pushing the boundaries of what is possible in embedded systems. As these devices become even more powerful and versatile, they will continue to transform industries and shape the future of technology.

NEXT MONTH'S TOPIC: IoT boards/modules Send related product announcements to editor@circuitcellar.com

Power Efficiency

The Power Efficient MCU is an integrated, ultra-low-power 32-bit microcontroller based on the ARM Cortex-M4F architecture, designed for industrial applications. Its features include a 384KB program memory, an integrated analog front end for low-power HART communication, and extensive I/O capabilities. The device excels in power efficiency, making it ideal for sensor and control systems. Advanced security features like secure boot and a built-in temperature sensor, the MAX32675C is tailored for demanding environments, ensuring reliability and performance in IoT and automation applications.

- ARM Cortex-M4F core
- Program memory size of 384KB
- Data memory size of 128KB
- Operating voltage range of 1.8V to 3.6V
- Ultra-low power consumption in active and sleep modes
- Integrated ADC with up to 12-bit resolution
- Multiple GPIO pins for flexible I/O configuration
- Support for I²C, SPI, and UART communication protocols
- Dimensions of 7mm x 7mm in a LGA-72 package
- On-chip temperature sensor
- Integrated watchdog timer for system reliability
- DMA controller for efficient data handling
- Support for PWM outputs
- Real-Time Clock (RTC) Enhanced security features

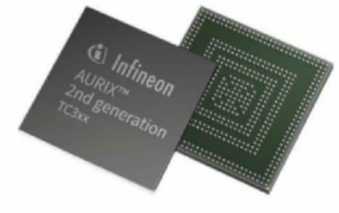
Analog Devices
www.analog.com

A Versatility

Designed for Versatillity, the Holtek HT32F12345 is a cutting-edge 32-bit microcontroller designed for high performance and low power consumption, built around the ARM Cortex-M3 processor core. It excels in versatility, making it suitable for a wide range of applications, from consumer electronics to industrial control systems. Key features include an efficient architecture, integrated digital and analog functionalities, and robust support for various communication protocols. This microcontroller stands out for its balance of performance and energy efficiency, making it an excellent choice for engineers looking to optimize their designs.

- ARM Cortex-M3 core
- Operating frequency up to 72MHz
- Flash memory capacity of 128KB
- SRAM capacity of 20KB
- Integrated ADC with up to 12-bit resolution
- Multiple PWM channels
- I²C, SPI, and UART communication
- GPIO pins with configurable functions
- Low power modes for energy efficiency
- On-chip oscillator and PLL
- Watchdog timer for system reliability
- DMA controller for efficient data transfer
- Integrated LCD driver support
- Temperature range from -40°C to +85°C
- Package options including LQFP and QFN
- Development support with Holtek IDE and libraries
- Comprehensive debugging features

Holtek
www.holtek.com



Tailored to Automotive Industry

The AURIX TC3xx series from Infineon is a state-of-the-art 32-bit microcontroller family designed specifically for automotive and industrial applications. It features a robust safety architecture, making it ideal for safety-critical environments. With up to six cores running at 300MHz, it offers exceptional performance and scalability. The series supports advanced functionalities such as data fusion and real-time processing, making it suitable for applications in autonomous driving and connected vehicles. Its compatibility with the previous TC2xx family ensures easy upgrades for engineers.

- TriCore architecture with up to six cores
- Max operating frequency of 300MHz
- Flash memory capacity up to 16MB
- Integrated RAM of up to 6.9MB
- Supports lockstep operation for safety-critical applications
- Enhanced security features
- Multiple communication interfaces including CAN and Ethernet
- Advanced safety features compliant with ISO 26262
- Low power consumption below 2W
- Pin compatibility with AURIX TC2xx family
- Integrated ADC and PWM functionalities
- Temperature range from -40°C to +125°C
- Extensive debugging capabilities
- Flexible power management options
- Support for real-time operating systems
- High-speed data processing
- Configurable GPIO pins for versatile applications

Infineon
www.infineon.com

DATASHEET URLS:

Analog Devices: www.analog.com/media/en/technical-documentation/data-sheets/max32675c.pdf

Holtek: www.holtek.com/webapi/116711/HT32F12345_Datasheetv140.pdf

Infineon: www.documentation.infineon.com/aurixtc3xx/docs/nho1702366509393

DATASHEET

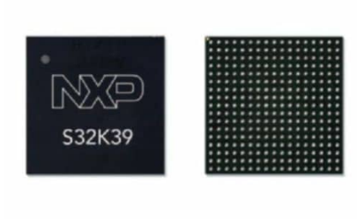
Datasheet: DSPs and DSCs

Advanced Graphics and Connectivity Features

The Microchip SAMA7D65 MPU, powered by a 1GHz ARM Cortex-A7 core, is specifically designed for Human-Machine Interface (HMI) applications. It integrates MIPI DSI and LVDS display interfaces, along with a 2D GPU, to deliver high-quality graphics and visual experiences. These high-performance features enable the transmission and processing of more data for efficient graphic performance, making it an optimal solution for HMI applications in industrial, medical, and transportation markets. Includes dual Gigabit Ethernet with Time Sensitive Networking (TSN) support, enabling precise synchronization and low-latency communication.

- Core Processor: ARM Cortex-A7
- Max CPU Speed: 1GHz
- Number of Cores: 1
- Bus Width: 32-bit
- L2 Cache: 256KB
- RAM Support: up to 8Gbit DDR3L
- Graphics Support includes 2D GPU, MIPI DSI, and LVDS
- I/O Interfaces include Dual Gigabit Ethernet and CAN-FD
- Package Type: 343-TFBGA
- Dimensions: 14mm x 14mm
- Operating Temperature ranges from -40°C to 105°C
- Security Features include AES, SHA, TDES, and TRNG
- Power Consumption: low power design for efficiency
- Development MPLAB Harmony 3
- TSN support for real-time networking

Microchip Technology
www.microchip.com



Designed for Electric Vehicles

The S32K39 microcontroller family from NXP is designed for advanced automotive and industrial applications, particularly in electrification. It features a powerful ARM Cortex-M7 core, capable of operating at 320MHz, which supports complex control algorithms and real-time processing. The S32K39 stands out with its high integration, offering up to 6MB of Flash memory and 800kB of RAM, making it suitable for demanding applications. Additionally, it includes robust safety features, extensive connectivity options, and support for AI/ML applications, ensuring versatility and reliability in various environments.

- Four ARM Cortex-M7 cores
- Maximum frequency of 320MHz
- Up to 6MB Flash memory
- 800KB RAM
- 237 I/O pins
- Supply voltage from 2.97V to 5.5V
- Integrated safety features
- Support for up to one six-phase or two three-phase motors
- Capable of handling control loops over 200kHz
- Advanced security features
- Multiple communication interfaces including CAN, LIN, and Ethernet
- Configurable as Lockstep pairs and Split-Lock cores
- Support for AI/ML workloads
- High integration for reduced system complexity
- Enhanced thermal performance
- Flexible power management options

NXP Semiconductors
www.nxp.com

High-Performance

The RAL41 High-Performance MCU family, particularly the RA4M1, is designed for high-performance applications, featuring an ARM Cortex-M4 core that operates at 48MHz. It stands out with its 256KB of flash memory and 32KB of SRAM, making it ideal for complex embedded systems. The inclusion of a segment LCD controller and a capacitive touch sensing unit enhances its usability in human-machine interfaces. Additionally, its low power consumption and versatile communication interfaces, including USB and CAN, make it suitable for a wide range of applications in consumer electronics and industrial automation.

- ARM Cortex-M4 core
- Maximum frequency of 48MHz
- 256KB flash memory
- 32KB SRAM
- 8KB DataFlash for EEPROM storage
- Segment LCD controller
- Capacitive touch sensing unit
- 14-bit ADC converter
- 12-bit DAC
- USB 2.0 full-speed module
- CAN communication interface
- Low power consumption features
- Multiple GPIO pins
- Integrated timers
- PWM outputs
- I²C and SPI interfaces
- Flexible power management options

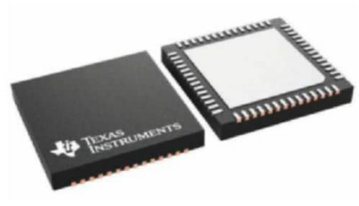
Renesas
www.renesas.com

Low-Power, High-Speed

The STM32H7 microcontroller series, featuring the ARM Cortex-M7 core, is designed for high-performance applications, operating at speeds up to 600MHz. It stands out with its dual-core architecture, combining the Cortex-M7 and Cortex-M4 for enhanced processing capabilities and efficient application partitioning. With up to 2MB of Flash memory and 1MB of RAM, it supports complex applications in automation, consumer electronics, and IoT. Its advanced peripherals, including high-speed connectivity options and low-power features, make it ideal for demanding embedded systems.

- Arm Cortex-M7 core
- Up to 600MHz operating frequency
- Dual-core option with Cortex-M4
- Up to 2MB of Flash memory
- Up to 1MB of RAM
- 192KB of TCM RAM
- Read-while-write support for Flash
- High-speed USB 2.0 interface
- CAN and Ethernet connectivity
- 12-bit ADC with multiple channels
- 16-bit DAC
- Flexible power management features
- Multiple GPIO pins
- Integrated timers and PWM outputs
- I²C, SPI, and UART interfaces
- Security features for data protection
- Advanced DMA controller for efficient data handling

STMicroelectronics
www.st.com



Real-Time Control Applications

The TMS320F280049 microcontroller from Texas Instruments is a powerful member of the C2000 family, designed for real-time control applications. It features a 100MHz C28x core with an integrated Floating-Point Unit (FPU), enabling efficient processing of complex algorithms. Its advanced peripherals include 16 ePWM channels for precise motor control and high-speed ADC capabilities, making it ideal for industrial automation and motor control applications. The microcontroller also supports low-power modes, enhancing energy efficiency, which is crucial for modern embedded systems.

- Core: C28x 32-bit
- Operating frequency: 100MHz
- Flash memory: 256KB
- SRAM: 100KB
- ADC 14 channels with 12-bit resolution
- ePWM channels total 16 with 150-ps resolution
- eCAP modules: 7 for enhanced capture capabilities
- eQEP modules: 2 for quadrature encoding
- Communication interfaces include I²C, SPI, and UART
- Low power modes are supported
- Operating temperature range: from -40°C to 125°C
- Security features are integrated for data protection
- GPIO pins offer flexible I/O options
- Trigonometric math unit is included
- IEEE 754 compliance is supported

Texas Instruments
www.ti.com

Motor Control Applications

The TMPM4KNF10AFG is a powerful 32-bit microcontroller from Toshiba's TXZ+ family, featuring an ARM Cortex-M4 core with a floating-point unit (FPU). It operates at frequencies up to 160MHz, making it suitable for demanding applications. With a code flash memory capacity ranging from 128KB to 1MB and data flash memory of 32KB, it offers ample storage for complex programs. Its low operating voltage range of 2.7V to 5.5V and robust performance at high temperatures (up to 105°C) make it ideal for industrial and automotive applications.

- ARM Cortex-M4 core
- Max operating frequency of 160MHz
- Code flash memory size from 128KB to 1MB
- Data flash memory size of 32KB
- RAM size ranging from 24KB to 64KB
- Operating voltage range of 2.7V to 5.5V
- Max operating temperature of 105°C
- Built-in FPU
- 32-bit data bus width
- LQFP package type
- Oscillation accuracy of less than ±1%
- Integrated peripherals for motor control
- Low power consumption features
- Advanced interrupt handling
- Support for various communication interfaces
- Development support through CMSIS Pack
- Enhanced security features for data protection

Toshiba
www.toshiba.com

DATASHEET URLS:

Microchip: ww1.microchip.com/downloads/aemDocuments/documents/MPU32/ProductDocuments/DataSheets/SAMA7D6-Series-Data-Sheet-DS60001851.pdf

NXP Semiconductors: https://www.nxp.com/docs/en/data-sheet/S32K39-S32K37-DS.pdf

Renesas: www.renesas.com/en/document/dst/ra4l1-group-datasheet?

DATASHEET URLS:

STMicroelectronics: www.st.com/resource/en/datasheet/stm32h743vi.pdf

Texas Instruments: www.ti.com/lit/ds/symlink/tms320f280049.pdf

Toshiba: www.toshiba.semicon-storage.com/eu/company/news/2025/01/micro-20250122-1.html

Modern Embedded Insights

Adopting Test-Driven Development in Embedded Systems

By
Jacob Beningo

Embedded software development has long been characterized by unique constraints such as limited resources, strict real-time requirements, and complex interactions with hardware. As a result, traditional software development methodologies often fail to translate smoothly into the embedded space. However, modern embedded development is evolving, and techniques like Test-Driven Development (TDD) are no longer just for web and enterprise applications: they're essential for building reliable and maintainable embedded systems.

COLUMNS

In my previous article, I introduced a modern CMake build system[1] designed to support flexible embedded development, including DevOps, simulation, and automated testing. Now, it's time to focus on one of the most powerful software engineering practices that this system enables: Test-Driven Development.

WHAT IS TEST-DRIVEN DEVELOPMENT (TDD)?

A big problem with software development is that developers write a lot of code quickly and then wait until the end to test it. Humans aren't perfect, and the chances are high that they won't catch every edge case and boundary condition. Making things worse is the fact that the testing is often done manually, so as the project grows in size, the time required to run those manual tests grows until there isn't enough time to test everything. The result is a system with bugs hiding and waiting to rear their ugly heads to users.

TDD is a development methodology that emphasizes writing tests before writing the actual implementation. Instead of blindly coding and hoping everything works at runtime, developers don't write a single line of production code unless a test is written to prove it does what it's supposed to do. There are many advantages to this approach. No code is written unless it's actually needed. All code has a test to prove that it works. Tests can be automated, which removes manual testing time

TDD enforces a disciplined approach to writing software. Developers use a simple process, which Kent Beck developed in his book Test-Driven Development. For embedded developers, though, the best interpretation of this list is the one that James Grenning put together in his book Test-Driven Development for Embedded C, called the TDD microcycle, which includes the following steps:

1) Add a small test
2) Run all the tests and see the new one fail, maybe not even compile
3) Make the small changes needed to pass the test
4) Run all the tests and see the new one pass
5) Refactor to remove duplication and improve expressiveness.

The TDD microcycle ensures that code is always written with a purpose, making debugging, maintaining, and scaling easier.

UNDERSTANDING THE TDD MICROCYCLE

The TDD microcycle revolves around rapid iterations of small tests and incremental development. Each loop should take no more than a few minutes, ensuring that code is continuously validated and refined.

Let's look at how this process works. Typically, when you are assigned a feature to develop, you create a list of tests necessary to assess the feature. The list won't be extensive. You'll likely come up with new tests as you develop the code. However, it helps a lot to sit down and list what tests you think you'll need to assess your new feature. Writing a list of tests is not included in the microcycle since it's something you do to set the stage for the microcycle.

Once you have your list of tests, you're ready for the first step: add a small test. You pick the most straightforward first test you could implement for your feature and write the test.

Next, run the test in your test harness. You might be thinking, "But Jacob, I haven't written the code yet!" That's precisely the point! You'd typically develop software by writing the code and then testing it. But how do you know that your test would catch a problem? You don't! You're leaving giant holes in your testing strategy when you develop your code first.

When you run the test in your harness before writing the code, you should see your test fail. In fact, you might not even see you code compile! The idea here is to see that the test case is implemented in a way such that it catches the code not working. That way, if it stops working in the future due to a code change, the test will catch it!

Now that we have a test case and have seen that it fails without the code, we're ready to start writing code. Make small, incremental changes towards implementing your feature. Don't go too far though! You want to write just enough code to make the test you wrote pass and nothing more!

Honestly, this is the toughest part when you are first adopting TDD. There is always a temptation to jump forward and write more code because you know what needs to be implemented. That is not the TDD way! Tests lead implementation.

Once you see that your test is passing, it's the perfect time to verify you haven't broken anything that already works. Rerun your entire test harness. Make sure the old tests and the new test pass with flying colors. Once you do that, you can be confident that your code is working as you expect it to.

Now, you can clean up your code, refactor, and set your eyes on repeating the cycle again by choosing your next test.

WHERE DOES TDD FIT IN EMBEDDED DEVELOPMENT?

Many embedded developers believe that TDD is difficult—if not impossible—to implement in their workflow. Traditional workflows rely heavily on debugging on hardware, which makes rapid iterations challenging. However, the modern approach to embedded software enables effective use of TDD by introducing off-target testing, simulation, and modular architectures.

TDD in embedded development typically fits into the following areas:

- Application Logic – Most business logic, data processing, and decision-making code can be tested without hardware dependencies.
- Device Drivers – While direct hardware access can be challenging, mocking frameworks allow developers to simulate interactions with registers, GPIOs, and peripherals.
- RTOS and Concurrency Management – Scheduling policies, thread synchronization, and inter-process communication can all be tested using TDD.
- State Machines and Control Algorithms – These often follow predictable patterns that lend themselves well to TDD-based testing.

The key to leveraging TDD isn't just adding a test harness to your builds. Using TDD in embedded development helps force you to decouple your application code from your hardware. It encourages you to use hardware abstraction layers. By following TDD you can more easily leverage simulation and integrate your software into a DevOps platform.

Let's look at how you can leverage TDD with the modern build system we discussed in the last time.

USING TDD WITH THE MODERN BUILD SYSTEM

My previous article introduced a modern embedded build system using CMake, supporting multiple build targets, including test and simulation builds. This infrastructure makes it possible to implement TDD efficiently in embedded projects.

In case you missed the last article, we used Docker to create a containerized build environment using GCC. That environment used a high-level makefile to allow to us build and run a Docker container. It also allowed us to build using CMake or using the high-level makefile. (Two build systems in one!).

Now, we'll look at how I integrated a test harness into that build system and how you can leverage TDD.

First, if you haven't done so already, you can download the modern build system example from the link in the resources section.

ABOUT THE AUTHOR

Jacob Beningo is an embedded software consultant and founder of the Embedded Software Academy, where he helps engineers master modern practices to design efficient, scalable embedded systems. He helps developers stay ahead in an evolving industry with hands-on training, expert insights, and strategies. He's consulted and trained companies such as Arm, General Motors, NASA, and others, completing more than 200 projects. Jacob has published books on embedded software development techniques. He holds three degrees, including a master's in space systems engineering from the University of Michigan. You can learn more about Jacob's work at www.beningo.com.

COLUMNS

Make sure that you have installed Docker and that you can run makefiles on your development machine. You'll also want to review the README file which describes how to make sure that everything is set up properly on your system. (There are small variations in the commands needed in the makefile based on whether you are on Windows or Linux-based systems).

From a terminal, you can build your docker container by running the command:

```
make docker_build
```

Once the container is built, you can run your container by executing the following command:

```
make docker_run
```

In order for us to run any tests, we need a test harness. A test harness is a collection of scripts, tools, and code that automate the execution of tests for a software application or system. It provides a controlled environment to execute test cases, capture results, and compare expected outputs with actual outcomes.

A test harness typically includes:

- A test execution framework to run test cases in a structured manner.
- Mocking/Stubbing components that replaces dependencies (e.g., hardware, databases) with simulated versions to isolate testing.
- Test data management that supplies predefined inputs for reproducible testing.
- Logging and reporting to capture test execution details, results, and failures.
- Automation capabilities that support continuous integration (CI) and regression testing.

There are several different test harnesses that are available for use in embedded systems that vary from open-source to commercial solutions. A few example open-source solutions include:

- CppUTest
- Unity
- Ceedling
- GoogleTest

The example build system that I put together uses the CppUTest harness. It is an open-source C/C++ test framework that is freely available and has most of the features you'd probably want in a test harness.

If you look inside the modern build systems makefile, you'll find that I've defined a recipe named unit-tests. The exact recipe looks something like the following:

```
.PHONY: unit_tests
unit_tests:
  $(MAKE) -j CC=gcc -f $(TEST_DIR)/cpputest.mk
  mkdir -p $(COVERAGE_DIR)
  @gcovr $(GCOVR_FLAGS)
```

You can see that it executes the cpputest makefile and then also runs an application named gcovr, which will tell you what your code coverage is. For example, if you have 100 lines of code in your application but the tests only execute 95 lines, then it will tell you that your test coverage is 95%. Not only will it tell you the coverage, it will tell you which lines weren't covered in the tests!

From your terminal, inside your docker container, if you run the command:

```
make unit-tests
```

You'll find that the CppUTest harness runs and provides you with the output seen in **Figure 1**.

You can see that this output actually has two different outputs. First, you can see the CppUTest output that tells you that there are 4 tests, 4 tests were run, there were 4 checks, and none of them were ignored. Eventually, as your tests grow in size, you won't want to run them all. You'll typically choose a subset and then let your CI/CD pipelines run the complete tests when you commit your code. The report enables you to know what was run and what wasn't.

Next, you can see that there is a GCC Code Coverage Report. The report tells you every file tested, the number of lines, those executed, and then the code coverage. As you develop your tests, it is very helpful to make sure that you have 100% coverage. However, be warned! 100% coverage does not mean you don't have any bugs! It just means that every line of code is being executed in the tests. You could have conditions in those lines that are not tested for and that are not correct!

```
Running firmware/tests/main_tests
....
OK (4 tests, 4 ran, 4 checks, 0 ignored, 0 filtered out, 0 ms)

make[1]: Leaving directory '/home/app'
mkdir -p firmware/tests/coverage
(INFO) Reading coverage data...
(INFO) Writing coverage report...
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
firmware/app/adder.c                           2       2   100%
------------------------------------------------------------------------------
TOTAL                                          2       2   100%
------------------------------------------------------------------------------
root@9343f3199ee1:/home/app#
```

FIGURE 1
CppUTest output on a simple application shows that fourl tests were executed with a 100% code coverage.

EXPLORING TDD

Let's dig a bit deeper into how this example works. You've seen how we run Docker and CppUTest, but let's look at some of the files and the tests that makes this example.

First, navigate into the firmware folder. You'll notice that the app folder contains an example application adder.c and adder.h. This module is an example of a feature that was added to the project. Next, look in the tests folder. You'll find the coverage report in the coverage folder. Feel free to explore it, but what we are really interested in is adder_tests.cpp.

If you open adder_tests.cpp, you'll find source code like that listed in **Listing 1**.

There are a couple of important features to notice. First, every test module that you create should have a setup and a teardown section. These allow you to refactor your tests and run the same code at the start of each test and at the end of the test. Code you might put in these areas would include things like:

- A reset for a state machine
- A reset for the state of a variable
- A way to clear a buffer

And so on. These sections help you to get your software into a good starting position to run your test.

Next, you'll notice the test cases. In our case, there are currently four of them. They look like the following:

```
TEST(adder, onePlusOneEqualsTwo)
{
    CHECK_EQUAL(2, add(1, 1));
}
```

Notice that they use a TEST macro that then takes two parameters: first the name of the test group, which for us is adder. Second, it takes a unique string name for our test. After that, the test itself is a series of macro calls that check the results of calling various functions in our module.

In the example above, we use the CHECK_EQUAL macro to make sure that if we call the function add with the values one and one, we get a result of two. You'll find that you can check the CppUTest documentation for a wide range of macros that can be used to validate your functions behaviors.

Now that you understand how the test harness works and a little about the TDD microcycle, I'd recommend that you try it out by doing the following:

1) Create a list of tests that would be needed for a subtractor module.
2) Add a new file to the test harness called subtractor_tests.cpp, and write your first test.
3) Run the test harness and watch it fail a test.
4) Add your subtractor module and write the minimum code to make the test pass.
5) Watch the tests pass.
6) Refactor.

Now you can repeat the microcycle for the remaining tests. By the end of the implementation, you'll have a much better feel for how you can use TDD for embedded development.

CONCLUSIONS

Test-Driven Development is not just a trend, it's a practical and essential approach for embedded developers looking to build robust, maintainable software. When combined with a modern build system that supports test and simulation builds, TDD becomes a powerful tool for accelerating development and reducing costly debugging time.

By following the steps outlined here and integrating CPPUTest into your build system, you can start leveraging TDD today to create better embedded systems faster. The key is to take it one step at a time—write a failing test, make it pass, then refactor. Before long, your test suite will become your greatest asset.

Once you get the feel for TDD, you'll find that it brings several important benefits to your development cycle:

- Better Code Quality
- Faster Debugging
- Reduced Time Spent on Hardware Debugging
- Easier Refactoring
- A flexible and scalable software architecture

Try out the example we've discussed in this article and see how you can unlock these benefits. When I first encountered TDDD I wasn't convinced. Today, I can't write a line of code without first writing a test!

```
#include "CppUTest/TestHarness.h"
extern "C"
{
  /*
   * Add your c-only include files here
   */
    #include "adder.h"
}
TEST_GROUP(adder)
{
    void setup()
    {
    }
    void teardown()
    {
    }
};
TEST(adder, onePlusOneEqualsTwo)
{
    CHECK_EQUAL(2, add(1, 1));
}
TEST(adder, zeroPlusZeroEqualsZero)
{
    CHECK_EQUAL(0, add(0, 0));
}
TEST(adder, OverflowByOneTest)
{
    CHECK_EQUAL(256, add(255, 1));
}
```

LISTING 1
Example test cases that were used to develop the adder module.

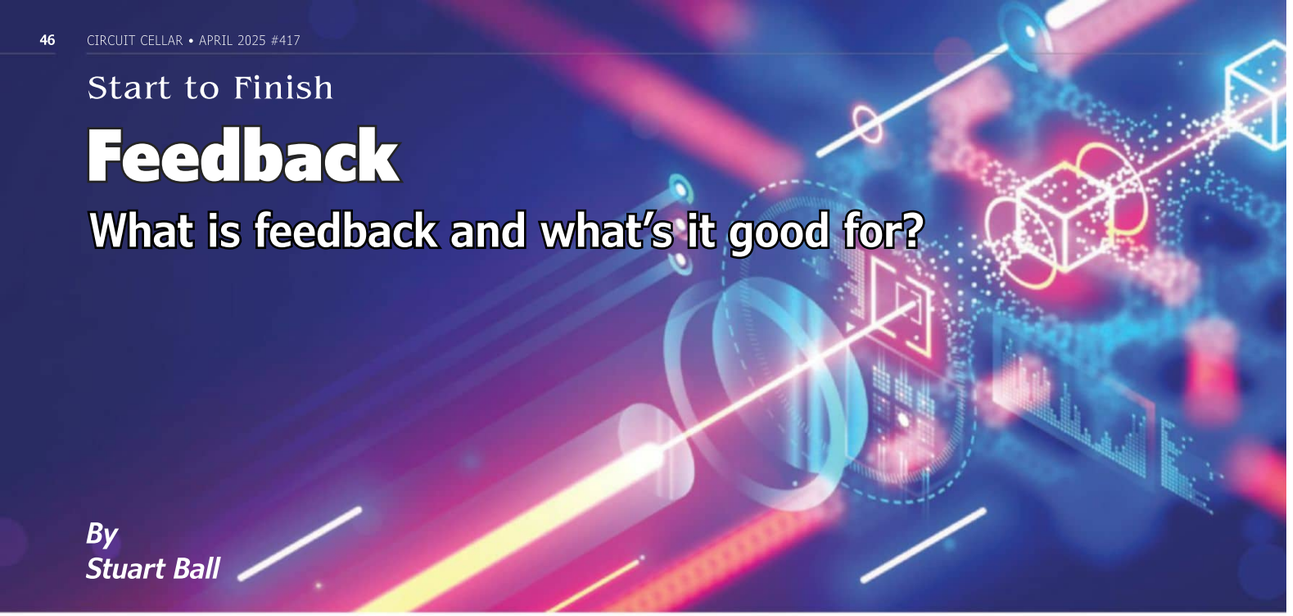
Additional materials from the author are available at:
www.circuitcellar.com/article-materials
Resources [1] as marked in the article can be found there.

COLUMNS

Start to Finish

Feedback

What is feedback and what's it good for?



By Stuart Ball

When most people hear the word "feedback" they probably think of the ear-piercing squeal that comes from a PA (public address) system when the microphone is too close to the speaker. But feedback is an essential part of electrical, software, and system design — as long as it's intentional and working how you want it.

WHAT IS FEEDBACK?

Feedback is what occurs when the output of a signal or system is fed back to an input, thereby affecting the output. Feedback comes in two basic varieties, positive and negative. Positive feedback is what hurts your ears in the aforementioned PA system squeal. With the exception of some rock bands, PA system feedback is usually something you want to avoid. Positive feedback occurs when the output is fed back to the input in-phase, reinforcing the signal.

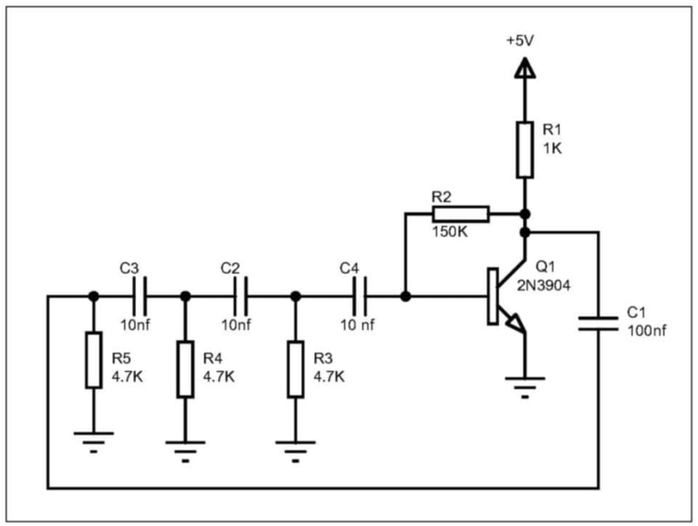


FIGURE 1
Phase shift oscillator schematic

POSITIVE FEEDBACK

Figure 1 is a schematic of a simple single-transistor phase-shift oscillator. This oscillator produces a sine wave at about 1.4KHz (depending on component tolerances) at the collector.

A phase-shift oscillator works because the resistor/capacitor ladder (R3, R4, R5, C2, C3, C4) produces 180° phase shift at one frequency and the inverting transistor produces another 180° phase shift, for a total of 360° phase shift. Provided there is enough gain in the entire loop, the circuit will oscillate. This circuit could also be built with an opamp. I won't go into oscillator theory here. I want to use this example to make a few points.

Figure 2 shows the startup characteristics of the oscillator. The lower trace is the 5V supply ramping up after power is applied (note the different voltage scale), the upper trace is the collector of Q1. Ignore the frequency display, the powerup ramp confuses the oscilloscope. You can see that the oscillation starts as the supply voltage approaches 5V, and then the amplitude slowly grows until it is oscillating between the supply rails. This is usually the limiting factor for this type of oscillator. It takes about 20 cycles for this oscillator to reach full amplitude.

The oscillator is started by electrical noise. In the PA system squeal, if you had the speaker and microphone in a very quiet room and slowly raised the volume, you could get it past the oscillation point before it started to squeal; the PA system needs some slight noise, maybe just hum in a speaker, to get the oscillation started.

All linear oscillators work essentially this way, whether the frequency is determined by a crystal, inductor/capacitor circuit, or by a resistor/capacitor network. Digital oscillators that operate by having three or more inverting stages with the output fed back to the input also need phase shift, but the phase shift is provided by the three inverters and is controlled by an RC network or simply by the propagation delay of the gates. I knew an engineer many years ago who said "if you want to build an oscillator, start by trying to build an amplifier". Which is not exactly true but illustrates the point that if there is enough gain and phase shift at a specific frequency, an amplifier will oscillate, even if it's unintentional. The circuit shown in Figure 1 is an amplifier, if you take out the feedback network.

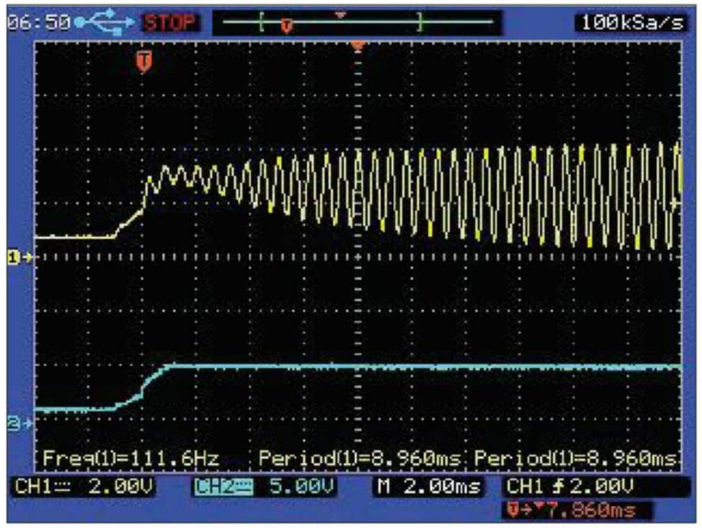


FIGURE 2
Oscillator startup waveform.

A HYSTERESIS SWITCH

Figure 3 shows a simple hysteresis switch built with two transistors. The output is high when the input is high, and the output is low when the input is low. The hysteresis switch is an example of useful positive feedback. Say the input is driven by a 0-1V sine wave, but we want to make it digital so we can connect it to a microcontroller (MCU) input. When the input is 0V, the output is also low, so the junction of R5 and the output is very close to zero. The voltage needed at the base of Q2 to cause the output to switch is about 0.6V. But R6 and R8 make a voltage divider, so the input has to be a bit over 0.6V, in fact about 0.62V to switch the output high. Now here's where the hysteresis magic comes in. Once the input goes high enough to drive the output high, R6 and R8 still make a voltage divider, only the junction of R8 and the output is now about 5V, so the input has to go lower than 0.6V to turn off Q2, in fact it has to go down to 0.4V.

The difference between the high and low switching states is 220mv, so if our sine wave was a little bit noisy, as long as the noise did not exceed 220mv, the switch would filter that out. The positive feedback alters the high and low switching points to convert the sine input into a digital output.

This isn't a very good hysteresis circuit, it's just to illustrate how hysteresis works. A comparator, with high impedance inputs, would make a better solution. But positive feedback in the form of hysteresis can clean up a noisy signal, up to a point. It will also speed up the switching time, although this transistor circuit is never going to be very fast.

BAD POSITIVE FEEDBACK

Of course, as the squealing PA amplifier proves, positive feedback can also be a very bad thing. If allowed to run unchecked, and if the amplifier has enough power, the speakers can be destroyed. Another example

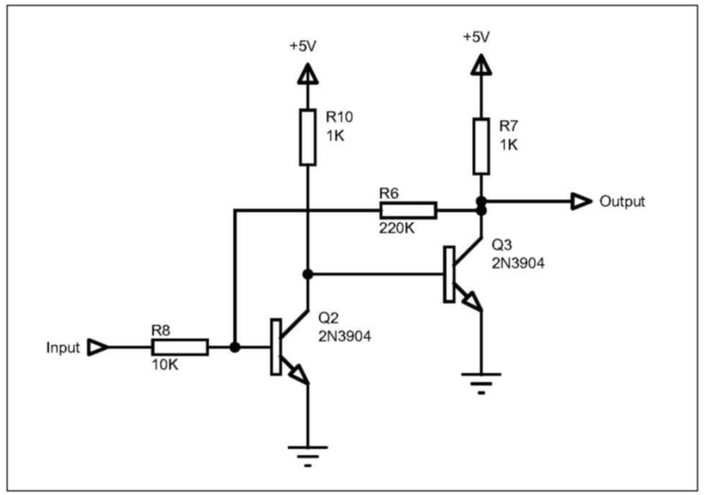
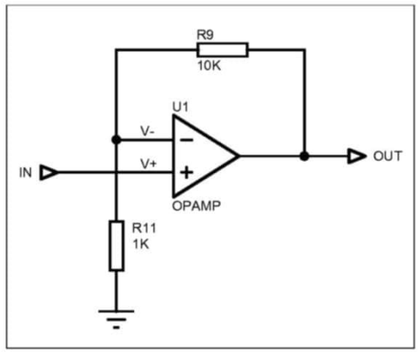


FIGURE 3
Hysteresis circuit.

FIGURE 4
Opamp feedback



of destructive feedback is thermal runaway. This term ordinarily applies to transistors, where transistor heating causes more power to be dissipated, causing more heat, and so on until the transistor fails. But thermal runaway can also apply to a circuit such as an embedded MCU controller. Say a system has a cooling fan that is controlled by the MCU, but the fan is stuck. As the heat builds up, the MCU tries to apply more and more power to the fan in an attempt to cool the system, but the added power just raises the heat more until something breaks. That's a contrived example, but it illustrates the point.

Positive feedback, both good and bad, applies to other things outside the world of electronics. For a dramatic example, search the Internet for the 1940 Tacoma Narrows bridge collapse.

NEGATIVE FEEDBACK

Positive feedback reinforces the output by feeding back the output to the input in-phase. Negative feedback does the opposite; the output is fed back to the input out of phase, which dampens the output. The phase-shift oscillator in Figure 1 has both positive and negative feedback: the positive feedback makes the circuit oscillate, while the negative feedback is how the transistor is biased.

Resistor R2 sets the bias point of Q1. A 2N3904 has a current gain (ß) of 100 to 300, meaning the collector current I_c is 100 to 300 times the base current I_b (I'm ignoring emitter current here). Ideally Q1 would be biased so that the collector voltage is around half the supply voltage when the feedback network is not connected, so we want the voltage at the collector of Q1 to be about 2.5V, which means 2.5mA through R1. But the key thing is that the bias point not put the transistor into cutoff or saturation. I_b is a function of the collector voltage (V_c), which is a function of I_c as follows:

$I_c \approx ß \times I_b$
$I_b = (V_c - 0.6)/R2 = (V_c - 0.6)/150K$
$V_c = 5 - I_c \times R1 = 5 - (I_c \times 1K)$
If you solve the simultaneous equations:
$V_c = (750K + 600\ ß)/(150K + 1000\ ß)$

So V_c varies from 2V to 3.2v over the range of ß from 100 to 300. Since I_b is a function of V_c, and V_c is a function of I_b, a lower gain transistor, near 100, will have a higher collector voltage, pushing more current into the base and bringing the collector voltage down. A higher gain transistor will have a lower collector voltage, providing less base current, so although the output is dependent on ß, using negative feedback compensates (partly) for ß variations and allows it to stabilize at a useful bias point through the range of possible ß values. Not necessarily a precise bias point, but one that is within an acceptable range. I measured the transistor I used in the example circuit, ß was just over 300, and with C1 removed (disabling the oscillator) the collector voltage was just over 2V. In this case, the negative feedback isn't compensating for variations in an input signal, but for device-to-device variations in transistor gain.

This is hardly the best way to bias a transistor for most applications. I chose this circuit to demonstrate that it's entirely possible to have both positive and negative feedback even in a simple circuit or system. In this case, the positive feedback enables oscillation through an AC path that operates at the oscillation frequency, and the negative feedback keeps the DC bias in a reasonable range so the oscillator will work. The positive and negative feedback don't interact in this circuit, although it would be possible to select DC bias component values that keep the oscillator from starting.

OPAMP EXAMPLE

Figure 4 shows a simple opamp circuit. The output of an opamp, assuming it is operating in a linear region (power supplies within range, output is not at either voltage rail), OUT = Av(V+ - V-). Av is the gain of the opamp, which is usually very high. In the figure, R9 and R11 provide negative feedback to control the gain. They are also a voltage divider between OUT and V-. So working through the math:

OUT = Av(V+-V-)
V- = (OUT x 1K)/11K = OUT/11
OUT = Av V+ - AV OUT /11
OUT/Av = V+ - OUT/11

Since Av is very large, the OUT/Av term is very small and can be discarded. So the resulting formula is OUT = 11 × V+.

If you've looked at a table of opamp configurations, you'll recognize this as a noninverting amplifier where the output voltage is R9/R11 + 1. Most opamp feedback calculations can be solved starting with these basic equations. In some cases, such as audio filters, you may have to take frequency into account, and if there is more than one voltage input (such as a voltage instead of ground at the lower end of R11) then the equations can get more complicated. But this is not an opamp tutorial, so I'll leave it at that.

In an opamp, if the V- input were simply connected to ground, the output would be Av V+. The LM2904 opamp, for example, has Av as high as 140V/mv, meaning each millivolt on V+ tries to move the output 140V. I say "tries to" because the opamp output won't go that high. Using negative feedback on the V- pin allows the gain to be precisely controlled.

This is the basic principle of any negative feedback system; the output is sampled and used to adjust the output value; if the output gets too high (whether that's voltage, current, speed of a fan, or any controllable value), the system adjusts the output to bring it back in line. The "system" can be a transistor, as in the phase-shift oscillator, an opamp circuit, or a MCU control circuit for a heater, fan, pump, motor, manufacturing process, or cruise control on an automobile. The basic concept is the same; on a cruise control, if the speed goes too low, the system applies more throttle to tell the engine to apply more power to the wheels. If the speed goes too high, the opposite happens. Of course, a cruise control system has many additional complexities and features, but the basic negative feedback is exactly the same as the opamp and transistor bias circuit; if the output is too high, make it lower. If it's too low, make it higher. That's negative feedback in a nutshell.

Control Methods: The actual process of applying negative feedback, especially for something like an industrial process, often involves "tuning" the system for maximum output or minimum variation or fastest correction. There are various schemes for doing this such as PID (Proportional-integral-derivative), PI (proportional-integral), proportional only, bang-bang control, predictive functional control, and others. This won't be a tutorial on control systems, that's a topic for another issue, and I looked at PID in the June 2019 Circuit Cellar issue. What I want to address here is the characteristics that affect the complexity and type of negative feedback to be used.

Cost/simplicity: A bang-bang control is the simplest, both in terms of control and the complexity of the controlled element. Most furnaces and air conditioners use bang-bang control. If you set the temperature in your house to 72°F, the heater or air conditioner doesn't turn partway on. It's either on or off. The negative feedback system senses the temperature, and when it gets to 72°F, it turns off. When the temperature moves away, the system comes back on. Bang-bang control is only suitable where there is some hysteresis in the system; when the temperature gets to 72°F, whether heating or cooling, it takes a while to ramp the other direction and turn the heat or air conditioning back on. Systems with no hysteresis or lag are unsuitable for bang-bang control because the controlled element will be constantly switching on and off. But the negative feedback principle is still applicable: when it gets too hot, make it cooler, and when it gets too cold, make it warmer.

Lag: Real-world negative feedback control systems have some lag. Even the opamp and transistor examples have some lag, but the input and output changes are too fast for the effect to be noticeable. I developed medical laboratory equipment many years ago. Let's use as an example a temperature control system in a medical device that is doing some processing on biological samples such as blood. The system has an MCU circuit that controls a heater to heat the sample cabinet to a specific temperature (also called a setpoint).

When the temperature goes low, the MCU will turn on the heater. When the heater reaches the setpoint, it is turned off. But any heater will take time to heat up. If it takes too long, the samples might get too cold, so the amount of power applied to the heater has to bring it up to temperature quickly enough to prevent that. Lag is the time it takes for the heater to actually do something when power is applied, or the time for liquid to flow through a pipe in an industrial control system, or the time for the engine to respond to a command from the cruise control in a car.

Overshoot: At the other end of the cycle, when the setpoint is reached, the heater is turned off. But it will remain hot for a while after the power is removed. This will continue to heat the cabinet, whether it's for seconds

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

ABOUT THE AUTHOR
Stuart Ball recently retired from a 40+ year career as an electrical engineer and engineering manager. His most recent position was as a Principal Engineer at Seagate Technologies.

COLUMNS

or minutes. This will cause the temperature to overshoot the target value, which may cook the samples or affect the process, so to avoid this overheating, the system might use a proportional-only control, where the drive to the heater is proportional to the difference between the setpoint and the actual temperature. The further the temperature is from the setpoint, the more drive is applied to the heater. As the temperature approaches the setpoint, the drive is reduced. You might do this manually when cooking: turn a stove burner very high to heat the water fast, and then as it is about to boil, turn down the heat so it doesn't splatter hot water everywhere.

Proportional control has two potential problems: the first is that if the proportional gain is too high, the temperature will overshoot the setpoint. A lower gain can prevent that, but it also causes slower response to changes. In addition, proportional control can cause the temperature to settle a little below setpoint because the difference isn't enough to activate the heater.

To fix this problem, an integral term might be added, making the system proportional-integral control. The integral term will activate the heater if it stays too long close-but-not-quite-at the setpoint, eliminating the offset. But the integral term also causes overshoot, which may have the same effect as the simple on-off control in overheating and damaging the specimens.

The point of all this is not to do a PID tutorial but to indicate what factors and considerations may be needed in a negative feedback control system. The negative feedback concept of feeding back the output to the input to adjust the setpoint to a fixed value is still valid. But how complicated that adjustment is varies from one application to another. Accuracy of the setpoint, speed of reaching the setpoint, amount of allowable overshoot or oscillation, and even the power efficiency are factors that may impact how the curve between current output and desired setpoint output is shaped. A poorly designed or adjusted control system can even result in positive feedback, causing oscillation.

SOFTWARE EXAMPLE

Figure 5 shows the schematic of a simple controller. This MCU PWM output drives an inductor to provide a boosted voltage of 10V, as you might use to power a sensor or opamp. If you saw my article in the February 2025 *Circuit Cellar* about switcher efficiency, you'll recognize this circuit. In this case, the output voltage is fed back to an analog-to-digital-converter (ADC) input on the MCU. This circuit was wired into a development board.

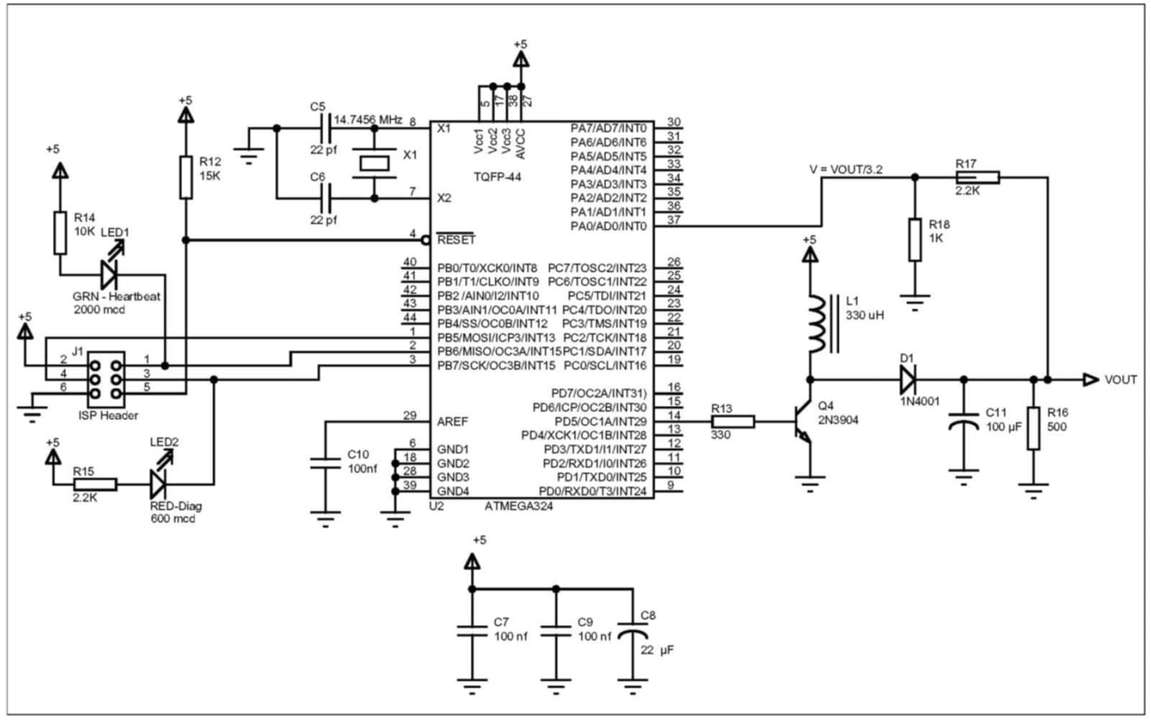


FIGURE 5
MCU controller schematic.

The example here is a single-loop control system. There are multiple-loop systems with multiple feedback points, and you can do a lot of math to model and simulate feedback systems, but that's beyond the scope of this article.

CONCLUSION

Both positive and negative feedback are important in circuits and systems. Control systems of all kinds are based on negative feedback. The difference between a simple transistor or opamp circuit and a sophisticated controller is the complexity of the negative feedback requirements and resulting control output waveforms. Designing a feedback system is like most of the hard decisions in life: you have to decide which set of problems you want to manage.

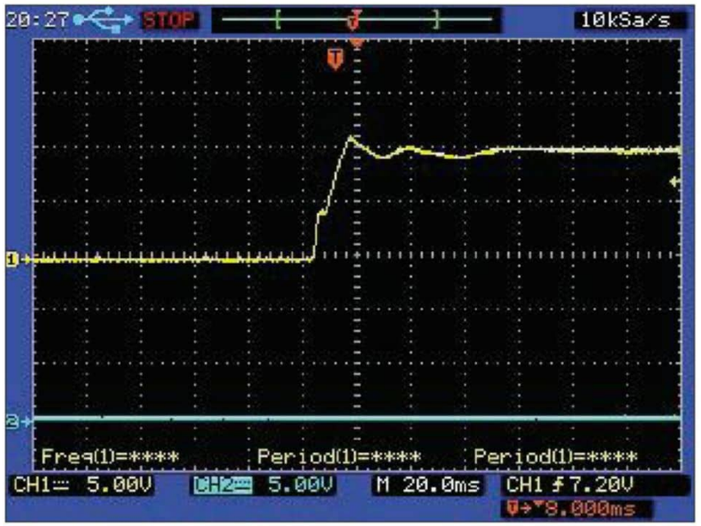
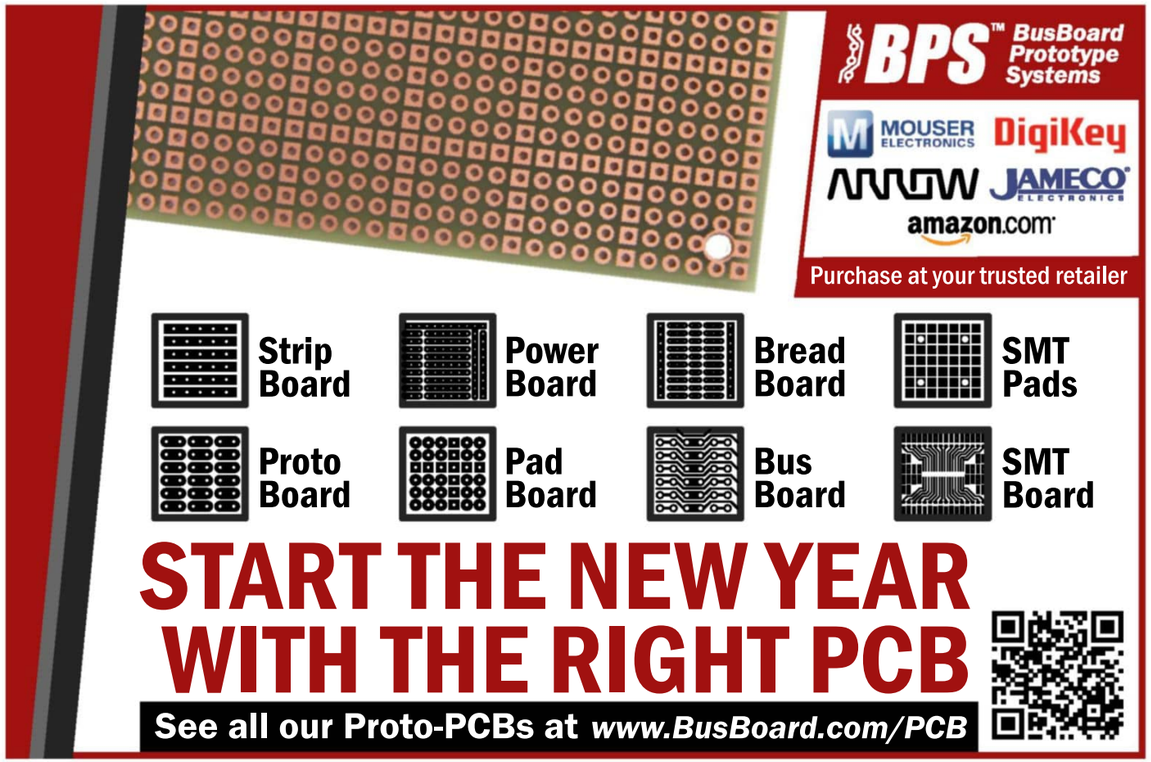


FIGURE 6
Proportional control waveform.



Embedded in Thin Slices

Introduction to Flutter State Management

By
Bob Japenga

As Bob continues this article series on Mobile App development from an Embedded System's perspective, he tackles State and State Management in Flutter. As you will see, it is very different from what embedded engineers are used to when we think of state and that misunderstanding can cause no end of trouble if we don't understand it.

COLUMNS

Over the years, I found that designing embedded systems using state machines to be a useful and powerful paradigm. There are different methods of implementing a state machine (Complete code for these is found in the Resources for this article): a table of function pointers indexed by the state (see **Figure 1** for a code snippet); the "switch-case" method (see **Figure 2** for a code snippet); or when using an object oriented language like C++, you can encapsulate each state in a class to represent the state and it's transitions (see **Figure 3**).

However, my hubris in thinking I was well versed in state machines did not prepare me for understanding state in either React Native or Flutter. As you may remember, I had two major problems with React Native: how it managed concurrency; and how it managed state. No surprise here. The same thing happened with Flutter, so let's start defining terms and see if we can unravel the mystery of states and state management in Flutter.

```
void main(void)
{
    State state = STATE_INITIALIZATION;
    while (1 == 1)
    {
        (*BottleMakingMachine[state].function)();
    }
}
```

FIGURE 1
Function pointers.

Everything in Flutter is made up of Widgets. And there are two kinds of Widgets: State less widgets and Widgets that change based on the state of the User Interface (UI). These are called appropriately StatelessWidgets and StatefulWidgets respectively. But wait! What do they mean by state?

REACTIVE/DECLARATIVE VS IMPERATIVE UI

Hold that question. Turns out that my experience in embedded systems and Windows development did not prepare me for Flutter states and state management. In Windows you build a screen out of components like radio buttons or sliders. Once built, if the component changes, you must update it. With Flutter (and React Native), the underlying framework builds the whole user interface all at once. You never just update a component (or Widget). Windows is by definition Imperative UI and Flutter (and React Native) are Reactive or Declarative UI's. I have been living in the Imperative paradigm (both in Windows development and in embedded systems development) and I have shifted gears without putting in the clutch.

NOW, LET'S TALK STATE

From Flutter's Reactive paradigm, there are two kinds of states: ephemeral states (also known as local or UI states) and app states. Ephemeral states are those that can be encapsulated within one widget. A checkbox is either in the state of on or off. Clicking the checkbox widget turns it on or off. But the state of a cart widget is more complicated because it is changed not by clicking on the cart but on clicking on other widgets (like an "Add to Cart" button). Thus, the state of a cart is an app state.

It is this connection between an "Add to Cart" button and the Cart that then requires some sort of management of state (what we call state management).

With all that aside, let's define state as used in Flutter:

State is information that can be read synchronously when a widget is built or information that might change during the lifetime of a widget. [1]

Perhaps you can see where my trouble started. State is being used in a very different way than when we created a Finite State Machine in our embedded systems.

The creators of Flutter (and React Native) wanted to decouple the screens from the underlying state and created a framework where the designer did not need to manually change the state of the UI every time the state changed. Thus, whenever you change the state of a given object, you must indicate to the framework that a state has changed and let it decide what needs to change.

As we will see in some examples in a bit, it is one thing to change the ephemeral state of an object. It is another to change the app state as in the cart example I mentioned. This requires some sort of state management to accomplish the state change. From the Flutter documentation on state management [2]:

State management is how we organize our app to most effectively access state and share it across widgets.

Notice the operative words: "share it across widgets." That's like the "Add to Cart" button sharing cart information with the Cart.

WHY DO WE NEED STATES

The question is often raised: Why do I need states? I haven't needed them in my web designs or in my embedded designs. That comes from core design decisions that the Flutter design team made in the beginning when they created StatefulWidgets. Let's look at an example you can run in Dartpad. (Just copy all the code from the main.dart in the Resources and place it in the Dartpad.com editor and click Run. NOTE: My apologies about putting all of the widgets in one file. A limitation of Dartpad! If you want to see the same code organized rationally in multiple files and multiple folders, check out the GitHub repository for this [3]). You should

```
void main(void)
    switch (state)
    case STATE_INITIALIZATION:
        state = STATE_WARMUP;
        break;
    }
    case STATE_WARMUP:
    {
        // Do work
        state = STATE_NORMAL_OPERATION;
        break;
    }
    case STATE_NORMAL_OPERATION:
    {
        // Do work
        state = STATE_SHUTDOWN;
        break;
    }
    case STATE_SHUTDOWN:
    {
        // Do work
        state = STATE_DORMANT;
        break;
    }
    case STATE_DORMANT:
    {
        // Do work
        state = STATE_DORMANT;
        break;
    }
    case STATE_ERROR:
    {
        // Do work
        state = STATE_INITIALIZATION;
        break;
    }
    default:
    {
        state = STATE_INITIALIZATION;
    }
    };
}
```

FIGURE 2
Switch case State Machine.

```
class LightSwitch
{
private:
    State currentState;
    std::unordered_map<State, std::unordered_map<std::string, State>> transitions;

public:
    LightSwitch() : currentState(OFF)
    {
        transitions[OFF]["flip"] = ON;
        transitions[ON]["flip"] = OFF;
    }

    void trigger(const std::string &event)
    {
        if (transitions[currentState].count(event))
        {
            currentState = transitions[currentState][event];
            std::cout << "Light is now " << (currentState == ON ? "ON" : "OFF") << std::endl;
        }
    }
};
```

FIGURE 3
C++ State Class.

COLUMNS

FIGURE 4
Main Screen.

Circuit Cellar: Embedded in Thin Slices
State Management Demo
Subscription: Discontinued
Circuit Cellar Reading Plan - No State Management
Circuit Cellar Reading Plan - InheritedWidget
Circuit Cellar Reading Plan - Provider

FIGURE 5
Screenshot from Android Phone.

4:02
Circuit Cellar: Embedded in Thin Sl...
State Management Demo
A Subscription: Discontinued
Circuit Celler Reading Plan - No State Management
Circuit Celler Reading Plan - InheritedWidget
Circuit Celler Reading Plan - Provider
Circuit Cellar Catalog of Issues to Buy
Circuit Cellar Simple Counter - using Provider

```
      child: ElevatedButton(
        style: style,
        onPressed: () {
          // Comment out the call to setState and the button will not update
          //setState(() {
          if (buttonEn) {
            buttonEn = false;
          } else {
            buttonEn = true;
          }
          //});
        },
        child: Text(buttonEn
            ? 'Subscription: Discontinued'
            : 'Subscription: Active'),
      ),
```

FIGURE 6
Using setState

```
      child: ElevatedButton(
        style: style,
        onPressed: () {
          // Comment out the call to setState and the button will not update
          setState(() {
            if (buttonEn) {
              buttonEn = false;
            } else {
              buttonEn = true;
            }
          });
        },
        child: Text(buttonEn
            ? 'Subscription: Discontinued'
            : 'Subscription: Active'),
      ),
```

FIGURE 7
Call to setState Commented Out.

see a screen like **Figure 4**. (Sorry again, but Dartpad only creates web apps, so it doesn't look like a phone. If you want to put it into Visual Studio Code and connect to a real phone, there are a lot more complications than we have time to cover in this short series. Trust me — this code runs on an Android and iOS phone. See **Figure 5** for how it looks on my Android phone in landscape mode.) Click on your Subscription for *Circuit Cellar* which has been "Discontinued". In the code, when the button is clicked, we change your status from Discontinued to Active (a good thing with your subscription to *Circuit Cellar*) by changing the text in variable `buttonEn`. But the Text doesn't change! Why? Because the framework hasn't been notified that the state changed via a call to `setState` (See **Figure 6**). Just changing the variable doesn't change the state of the widget. Uncomment the call to `setState` (See **Figure 7**) and see how the button now works properly. That is why, in the Flutter framework you need states.

Why do I need State Management? Since you cannot change a Widget from outside (for example, your "Add to Cart" button cannot just call the `IncrementCart` function in the Cart widget), you need some scheme for you to communicate to other stateful widgets. And that is done with state management. Let's again illustrate this with an example from our code.

If you click on the *Circuit Cellar* Reading Plan with No State Management button, you will see a scrollable list of articles from *Circuit Cellar* to read (See **Figure 8**). Of course, Embedded in Thin Slices is at the top! Once you have read the article, just click on the article and it will get checked off. But wait! The progress bar is not updated. Because, like the cart, it is another widget and there is no way to change the widget from outside without state management. Interesting behavior when you go back to the main screen, and then back to our No State Management screen, the progress bar is updated. That is because the progress bar is re-built from existing state data.

STATE MANAGEMENT TOOLS

Flutter has provided a few state management specialized widgets to provide this functionality:

InheritedWidget: With this widget, you can define a number of objects that can be accessed by other widgets in the widget tree. In our example code, we defined a few: an integer - `numberRead` and a function - `updateProgress`. When the checkbox is clicked, the `numberRead` is updated by calling `updateProgress`. And because these are inherited by all the widgets under the widget tree, the progress bar is updated. Voila! State management. When the state of one widget changes, another gets notified. In our Dartpad application, you can click on the "*Circuit Cellar* Reading Plan – InheritedWidget" button to see that the progress bar now works.

InheritedNotifier: Perhaps the easiest way to distinguish this from the InheritedWidget is that the widget is notified when a value changes where as InheritedNotifier widgets are notified on demand. For example, in my MP3 streaming app, the progress bar does change when the position in the MP3 changes (it is changing continuously) but when I notify it that it changes – for example every second instead of every microsecond.

InheritedModel: Imagine that you had a data structure that was recording data from a number of sensors in an avionics package. You wouldn't want the altimeter widget to get updated every time that the air speed changed. To accomplish this, the Flutter design team created the InheritedModel widget. With this you can take just some of the data, and limit who gets notified.

HOUSTON! WE HAVE A PROBLEM

Those and a few other state management tools were provided by the Flutter team. But if you go to the Flutter documentation by the Flutter team about state management they give you examples using other state management tools like: Provider and Riverpod. Why is that? I won't get into the "state management wars" in cyber space, but they are fierce. I attempted to use Riverpod in my app and quickly got snowed. It was way too complicated for a bear of very little brain like

FIGURE 8
No State Management.

SIGLENT®
And
Saelig
unique electronics
Present the
SDS800X HD
HD Digital Storage Oscilloscopes
From
$339
• 70 MHz, 100 MHz, 200 MHz bandwidth models
• 12-bit high resolution
• Real-time sampling rate up to 2 GSa/s
• Memory depth up to 50/100 Mpts
• Waveform capture rate up to 500,000 wfm/s (Sequence mode)
• 7-inch touch screen, support for external mouse and keyboard
• www.saelig.com • 585-385-1750 •

me. But it seems to be the most popular state management paradigm. Provider is supposed to be a little higher level than InheritedWidget and easier to learn than Riverpod. So that sounded good to me. In the example code you will see the option to plan your *Circuit Cellar* Reading using Provider state management (click on the "*Circuit Cellar* Reading Plan – Provider" button). Let me briefly summarize how Provider works.

PROVIDER

Given the dissatisfaction with the Flutter provided state management, an organization that calls themselves dashoverflow created the Provider state management package[4]. It is intended to be a higher level wrapper around the InheritedWidget and InheritedNotifier mechanisms provided. With Provider, you create a model for each object that you want to provide state management. For example, in our example code, we have a model for the checkbox (See **Figure 9**). This has a toggle function and an array of checkboxes and it has a way to increment and decrement the total articles read. Both the Progress widget and the Checkbox widget access these objects through the widget's context. You can set up an object to be watched and be notified if it changes or you can set it to read the object. Remember when I said that the innards of a widget cannot be changed from the outside? With Provider they can be.

```
class CheckboxModel with ChangeNotifier {
  final List<bool> _checkbox = List.filled(22, false);
  List<bool> get checkbox => _checkbox;

  void toggle(int i) {
    _checkbox[i] = !_checkbox[i];
    notifyListeners();
  }
}

// This class is used for counting the number of articles read
class ReadCountModel with ChangeNotifier {
  int _count = 0;
  // This is the getter function count which returns _count
  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }

  void decrement() {
    _count--;
    notifyListeners();
  }
}
```

FIGURE 9
Models for provider.

A PLETHORA OF OPTIONS

The older I get, the less I like too many options. The official Flutter documentation points you to no less than 15 different outside packages for Flutter state management. How is that possible? Because everyone thinks they have a better way. All are doing basically the same thing. This creates a mess for re-using code. For my purposes in my Sermon Engagement app, my need for state management is with the audio player. The audio player has a progress bar widget that needs to be updated from several places: the internal player as the MP3 plays; the slider on the screen to move the progress bar; and the Bluetooth headset which changes the progress with fast forward and skip. I avoided state management tools with React Native by using (horror of horror) globals. React Native allowed that. But I couldn't get the same global implementation to work with Flutter.

The package that supported Bluetooth headset control used one flavor of state management and the package that provided example code for the progress bar used Riverpod. I attempted to use them, but found that trying to blend them I ended up with mud, and not-working mud at that. This is where re-usability suffers when the core design doesn't support a basic feature in a robust way.

CONCLUSION

Something tells me that the Flutter development team dropped the ball. They created a framework that required state management but they provided something that proved to be inadequate by the user community.

My recommendation is to avoid state management at all costs wherever possible. Keep It Simple Smarty.

Next time, we will release this puppy to the App and Play stores. Or at least try to in thin slices.

ABOUT THE AUTHOR

Bob Japenga has designed embedded systems since 1973. From 1988-2020, Bob led MicroTools, an engineering firm specializing in creating a variety of real-time embedded systems across a wide range of industries. Bob has been awarded 11 patents in many areas of embedded systems and motion control. Now retired, he enjoys building electronic projects with his grandchildren. You can reach him at Bob@ListeningToGod.org.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [4] as marked in the article can be found there.

By Kirsten Campbell **PRODUCT NEWS**

Infineon CoolGaN Power Transistors Enable SounDigital to Reach Higher Fidelity in Smaller Amplifier Systems

Manufacturers of cutting-edge audio equipment constantly seek to enhance sound quality while also meeting the growing demand for compact, lightweight, more integrated, and energy-efficient designs. At the same time, they must ensure seamless connectivity, cost-effectiveness, and user-friendly functionality, making audio product development more complex than ever. To overcome these challenges, SounDigital has integrated CoolGaN transistors from Infineon Technologies AG into its new 1500W Class-D amplifier, featuring an 800kHz switching frequency and five channels. Infineon's advanced GaN technology has improved the energy efficiency of the amplifier by 5% and reduced energy loss by 60%.

"We are excited to enhance the performance of our audio amplifiers using Infineon's GaN power semiconductors, enabling us to inspire people and provide entertainment by amplifying music around the world," said Juliano Anflor, CEO of SounDigital. "GaN transistors significantly enhance our overall system performance with minimized system cost and increased ease of use."

"GaN technology is transforming the audio amplifier industry, providing unparalleled efficiency and performance," said Johannes Schoiswohl, Head of the GaN Business Line at Infineon. "Infineon's leading GaN solutions deliver superior sound quality, higher power density, and reduced energy consumption, enabling SounDigital's audio systems to reach new levels of fidelity and performance."

For its 1500W Class-D amplifier, SounDigital selected Infineon's 100V normally-off E-mode transistors: IGC033S101 in a PQFN-3×5 package and IGB110S101 in a PQFN-3×3 package. With their low on-resistance, the transistors are ideal for demanding high-current applications, enabling significant improvements in both sound quality and efficiency of SounDigital's amplifier.

The GaN-based amplifier also delivers high performance while reducing power dissipation by 75W, allowing for a 50% smaller heatsink. Additionally, the overall system size has been reduced by 40% without compromising performance. The audio quality has been further improved by the CoolGaN transistors, with total harmonic distortion (THD) reduced by 70%, enabling a more precise and detailed sound experience. At the same time, the idle current has been reduced by 40%, significantly improving energy efficiency.

Infineon | infineon.com

Quectel Unveils FGM842D Series of Wi-Fi and BLE Modules for Smart Homes and Industrial IoT

Quectel Wireless Solutions, a global IoT solutions provider announces the Quectel FGM842D and FGM842D-P Wi-Fi 4 and Bluetooth Low Energy 5.2 modules. The modules feature high performance ARM968 processors with built-in 288KB RAM and 2MB flash memory to ensure efficient performance. The modules support UART and GPIO interfaces and offer support for SPI, I2C, ADC, and PWM functions in the QuecOpen solution, which offers various low power consumption modes and a keep-alive mechanism to deliver maximum flexibility to designers and developers.

Ideal for smart homes and industrial IoT deployments, the modules comply with the WPA-PSK, WPA2-PSK and WPA3-SAE security standards and support True Random Number Generator (TRNG) and the AES-128 encryption algorithm. In addition, the modules feature secure firmware and flash encryption.

"We're delighted to launch the Quectel FGM842D and FGM842D-P," said Norbert Muhrer, President and CSO, Quectel Wireless Solutions. "These compact, high-performance Wi-Fi 4 and Bluetooth Low Energy 5.2 modules offer optimized size and cost and are extremely versatile. This makes them suitable for a wide range of use cases across smart homes, industrial IoT and many other deployment scenarios."

Broadening their appeal further, the modules have a compact LGA package, measuring 12.5mm × 13.2mm × 1.8mm for the Quectel FGM842D and 16.6mm × 13.2mm × 1.8mm for the FGM842D-P. The FGM842D-P features a PCB antenna rather than the FGM842D's third-generation RF coaxial connector or pin antenna interface. Operating in the 2.4GHz Wi-Fi frequency band, the modules operate in temperatures ranging from -40°C to +105°C.

Quectel | quectel.com

NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

STMicroelectronics' New Integrated STM32WBA6 Wireless Microcontrollers Combine Extra Features and Performance with Power Efficiency

STMicroelectronics, a global semiconductor leader serving customers across the spectrum of electronics applications, has announced the next generation of its STM32 power-efficient short-range wireless microcontrollers (MCUs) that simplify connecting consumer and industrial devices to the IoT.

The new STM32WBA6 series is used in connected, smart devices such as wearable healthcare and wellness monitors, animal collars, electronic locks, remote weather sensors, and more. Packing extra memory and digital system interfaces, while preserving energy efficiency, the new MCUs can handle richer functionality in emerging new product designs.

The STM32WBA6 MCUs also embed SESIP3 and PSA Level 3 certifiable security assets, such as cryptographic accelerators, TrustZone isolation, random generator, and product lifecycle that will contribute and enable ST customers to reach compliancy toward the upcoming RED and CRA regulations.

"Robust and standardized wireless connectivity is central to the IoT's success. Our new STM32WBA6 MCUs bring richer features and larger memory to address high-end applications in smart home, health, factory, and agriculture," said Patrick Aidoune, General-Purpose MCU Division General Manager, STMicroelectronics. "Our customers can now increase the pace of development to meet demands from consumer and industrial markets for new products that deliver more features and increased capabilities within reduced size and power constraints."

The wireless subsystem in the new STM32WBA6 microcontrollers supports Bluetooth, Zigbee, Thread, Matter, and other protocols operating in the 2.4GHz frequency band, and allows communication using multiple protocols concurrently. It's how a system such as a smart-home bridge can communicate with the homeowner's mobile app over Bluetooth and simultaneously manage lights or thermostats through mesh networking such as Zigbee. The STM32WBA6 series also contains single-protocol variants for simpler and more cost-conscious applications.

STMicroelectronics | st.com

PRODUCT NEWS

Nordic Semiconductor and Skylo Partner to Bring Ultra-Low Power Satellite Connectivity to Massive IoT

Nordic Semiconductor, a global leader in low power wireless connectivity solutions, and Skylo, the pioneer in Non-Terrestrial Network (NTN) communications, are announcing a strategic partnership to certify Nordic's nRF9151 low-power cellular module on Skylo's satellite network service. Nordic and Skylo's partnership unlocks new massive IoT use cases, such as remote monitoring, asset tracking, and enhanced safety and security by enabling small, constrained IoT devices to connect seamlessly over satellite. It marks a clear path for device manufacturers to achieve global connectivity using 3GPP-compliant, off-the-shelf cellular solutions.

The award-winning ultra-compact nRF9151 is the lowest power cellular IoT solution on the market today with industry-leading battery lifetime performance. The nRF9151 module is globally certified and features support for both LTE-M, NB-IoT, and DECT NR+ modem alongside an open application MCU core, with GNSS and cell-based location capabilities. Skylo device certification and NTN support will be announced in a product update later this year, bringing even more connectivity options to users.

"By leveraging Skylo's existing, commercially available satellite network, we are today enabling connectivity anywhere," said Oyvind Birkenes, EVP of Nordic' Long Range Business Unit. "This collaboration with Skylo opens the door to a new class of IoT solutions that can operate in remote or off-grid environments with standard ultra-low power cellular solutions like the nRF9151."

"Partnering with Nordic exemplifies the kind of connectivity innovation we are proud to support at Skylo," said Tarun Gupta, co-founder and Chief Product Officer at Skylo. "We see strong demand from the market for satellite connectivity on low power modules such as the award-winning nRF9151. This will enable device manufacturers to create innovative new products that stay connected far beyond the limits of terrestrial coverage. It's a significant step toward our vision of ubiquitous, seamless coverage for all devices, and we're excited to see the new massive IoT applications that will now be made possible from this partnership."

With this partnership, industrial, consumer, and healthcare markets can now benefit from ultra-low power satellite connectivity. Small, battery-operated devices—from environmental sensors deployed in remote locations to trackers outside standard terrestrial connectivity—can stay connected longer, leveraging Skylo's standards-based 3GPP satellite network, alongside Nordic's nRF9151 module. This advancement enables seamless global coverage without reliance on terrestrial networks, unlocking new possibilities for massive IoT applications.

Nordic Semiconductor | nordicsemi.com

NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

IDEA BOX

The Directory of PRODUCTS & SERVICES

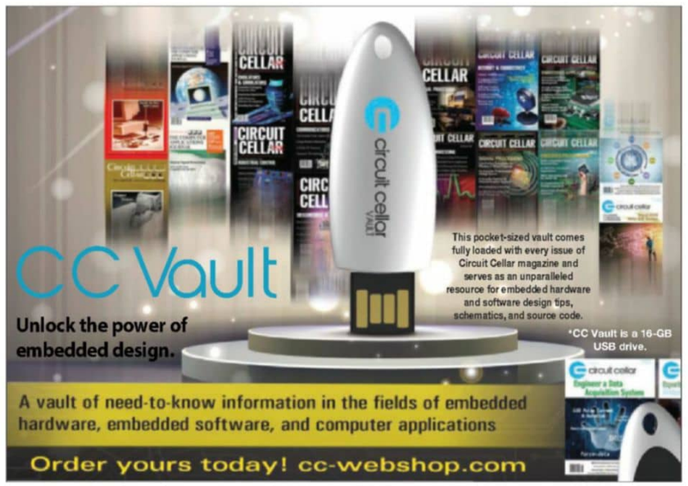
AD FORMAT:
Advertisers must furnish digital files that meet our specifications (www.circuitcellar.com/mediakit).

All text and other elements MUST fit within a 2" x 3" format.
E-mail adcopy@circuitcellar.com with your file.

For current rates, deadlines, and more information contact Hugh Heinsohn at 757-525-3677 or Hugh@circuitcellar.com

Verilog HDL
With the right tools
designing a microprocessor can be easy.
Okay, maybe not easy, but certainly less complicated. Monte Dalrymple has taken his years of experience designing embedded architecture and microprocessors and compiled his knowledge into one comprehensive guide to processor design in the real world.
Monte demonstrates how Verilog hardware description language (HDL) enables you to depict, simulate, and synthesize an electronic design so you can reduce your workload and increase productivity.
cc-webshop.com
Microprocessor Design
Using Verilog HDL

RAPID18
Universal Development Kit for PIC18 Family
ONLY $99!
• Real-time Clock/Calendar with Supercap
• USB port for text communication to the running program
• Built-in bootloader for loading code
• Excellent for AC/DC Data Logging
Kit includes everything you need to develop with a PIC® MCU:
• Rapid 18 Prototyping Board
• Single-Chip IDE C Compiler
• Exercise Book
• Power Adapters and Cables
sales@ccsinfo.com (262) 522-6500 x35
www.ccsinfo.com/cc1223


CC Vault
Unlock the power of embedded design.
This pocket-sized vault comes fully loaded with every issue of Circuit Cellar magazine and serves as an unparalleled resource for embedded hardware and software design tips, schematics, and source code.
*CC Vault is a 16-GB USB drive.
A vault of need-to-know information in the fields of embedded hardware, embedded software, and computer applications
Order yours today! cc-webshop.com

embeddedTS
embeddedTS.com
TS-7100-Z
Powered by the high performance, ultra-efficient Arm® Cortex®-A7 based i.MX6UL CPU
...In an enclosure as small as two decks of cards
(2.4" x 3.6" x 1.7")
Best In Show
Embedded Computer Design
Made in the USA
with global parts

TEST YOUR EQ

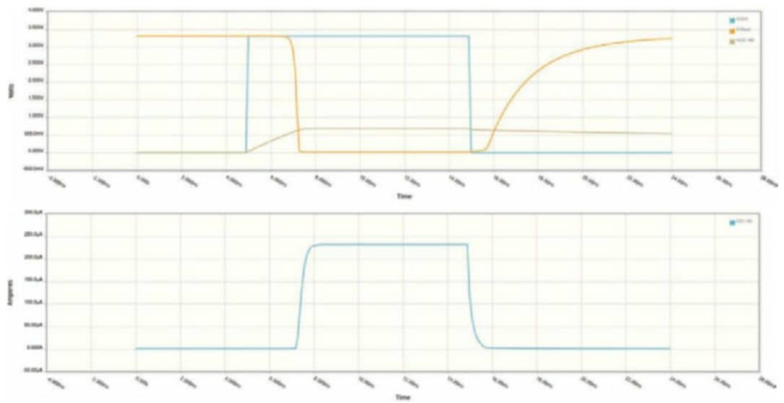
Contributed by David Tweed

Answer 1 —

The output will switch high-to-low when Q1's base voltage reaches the point where the base current is enough to saturate the collector, at about 0.7V. When the input switches to 3.3V, the capacitor is charging from a Thévenin source of 3.0V, as determined mainly by R1 and R3. The time required to reach 0.7V is therefore approximately -ln(1 - 0.7/3.0) = 0.266 times the R-C time constant, or about 2 ms.

Answer 2 —

That's actually a much trickier question. The output will switch low-to-high when Q1's base voltage drops to the point where the base current is no longer enough to saturate the collector. This point is very sensitive to the characteristics of both the transistor (gain, internal resistance) and the load that it is driving — the 0.3mA flowing through R4 in this case. Simulation shows that for this particular configuration, the delay in this case is also close to 2ms, or about 0.04 times the R-C time constant (as determined by R1, R3 and C1). This is nowhere near the 10ms (factor of 5) one might expect at first glance.

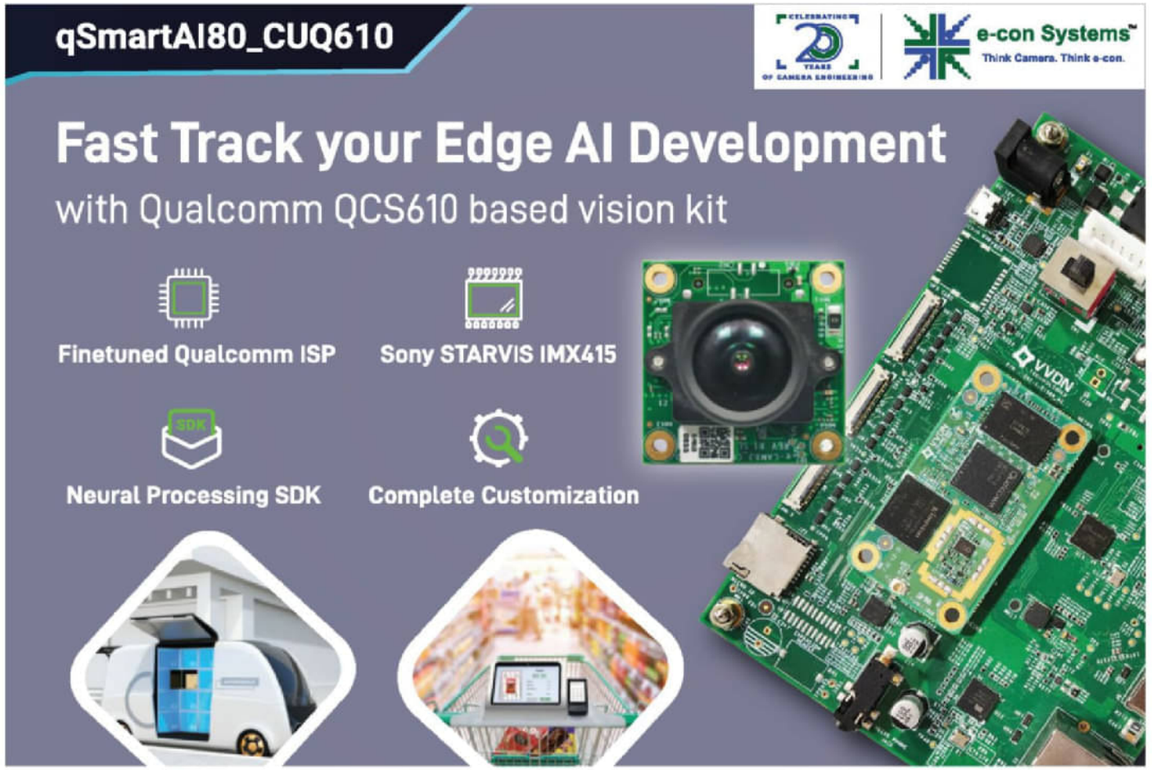


Answer 3 —

The fundamental problem is that the B-E junction of the transistor clamps the capacitor voltage at about one diode drop, or 0.7 V. This in turn means that the turn-off timing of the circuit is highly sensitive to the characteristics (mainly current gain) of the transistor.

Answer 4 —

The simplest change would be to add a resistor in series with the base of the transistor, to limit the base current and allow the capacitor voltage to rise to a higher value. This would make the turn-off timing of the circuit much more predictable.



The Future of AI Integration

What It Really Takes for a General-Purpose MCU to Function as a Complete AI SoC

Innovative AI-enabled MCUs are transforming endpoint devices by integrating advanced AI capabilities while addressing critical challenges in power and space. This article explores the future of wearable technology and the engineering solutions that make it possible.

By
Mark Rootz
Vice President of Marketing, Alif Semiconductor

The application of AI to endpoint devices can transform their value. In the medical arena, wearable AI devices promise to take the detection and diagnosis of serious conditions such as atrial fibrillation out of the clinic and into the patient's everyday world for monitoring. And by applying AI algorithms, hearing aids can be transformed from a simple amplifier into an intelligent vocal discriminator, isolating the voice of the person of interest, while cancelling all other noise and voices, or muting them in the sonic background.

These AI capabilities could multiply the value of almost every type of wearable and portable battery-powered product. And at the endpoint, AI functions can often not be performed in the cloud for reasons of power, latency, privacy, wireless reach, security and cost. Such devices need local AI processing capability.

But the successful local deployment of AI in these products must find a way past the severe design constraints of space and power. Wearable devices such as earbuds, rings, smart glasses, and patient monitors have small enclosures which can accommodate few components and only a small battery.

Pre-AI, many of these types of products (although endowed with fewer features and physically larger) were based on general purpose microcontrollers or microprocessors which could integrate the main functions. Integration helps the designer meet their goals for space and power while reducing component count and board footprint.

In the AI era, integration of system functions into a general-purpose MCU remains a highly desirable way to save space and power. But an AI MCU must integrate even more functions than earlier conventional MCUs, while at the same time providing the ultra-low power consumption that enables devices with small batteries to run AI at the endpoint without sacrificing run-time between charges.

Cortex-M55 and Ethos-U55 256MAC at 400MHz, Ensemble High-Performance Subsystem All Data Shown is for a Single Inference Only										
ML Model	**NPU + CPU**				**CPU Only**				**Improvement**	
	Time ms	Power mW (Δ)	Current mA (Δ)	Energy mJ (Δ)	Time ms	Power mW (Δ)	Current mA (Δ)	Energy mJ (Δ)	Inference Time Reduction	Energy Reduction
KWS[1]: MicroNet Medium (ARM)	6.8	25.7	7.6	0.17	137	19.3	5.7	2.62	20x	15x
Object Detection[2]: YOLO-Fastest (face trained)	7.3	33.8	10.0	0.25	657	21.3	6.3	13.7	90x	55x
Image Classification[3]: MobileNet v2	20.1	43.3	12.8	0.86	2707	24.0	7.1	62.4	135x	73x
Auto Speech Recognition[4]: Tiny ASR (Wav2letter)	27.9	29.6	8.9	0.89	4534	17.3	5.2	77.8	163x	87x

FIGURE 1
Benchmark tests show the superior performance and efficiency of an NPU over a CPU when executing common ML functions.

This is not as simple as bolting AI capability onto a legacy MCU architecture. Alif Semiconductor, founded in 2019 to create a new generation of AI-enabled MCUs for endpoint devices, has had the advantage of thinking from first principles about how to integrate AI into an MCU. This thinking reflects the hundreds of conversations in which OEMs have described to Alif the factors that affect their choice of MCU to serve as an AI system-on-chip (SoC) for battery-powered endpoint devices. Here are four of the most important such factors.

NEURAL PROCESSING ACCELERATION MUST BE TIGHTLY COUPLED TO THE CPU

The MCU market's initial response to customer demand for AI functionality was to develop AI software development kits (SDKs) to enable AI/machine learning (ML) algorithms to run on the same Arm Cortex-M CPU that also performed conventional control functions. However, an MCU for endpoint ML applications needs a neural processing unit (NPU): this type of processor is optimized for the multiply-accumulate (MAC) operations that are the bread and butter of neural networking applications. An embedded CPU on its own will struggle with meaningful ML workloads because inferences derived from the highly parallel ML network must be resolved in a serial fashion, taking excessive time while burning a lot of energy.

Figure 1 illustrates the contrast in AI performance between an MCU's CPU and NPU. Alif Semiconductor's Ensemble MCU family employs the contemporary Cortex-M55 CPU core paired with an Ethos™-U55 NPU co-processor. The metrics show a single ML inference for four trained ML models running on an Ensemble MCU: the models represent the operations required for keyword spotting, object detection, image classification, and speech recognition.

The Cortex-M55 used in Alif MCUs is among the best of embedded CPU architectures, and puts up good numbers which are already on the order of 5x better for ML workloads than previous generations of Cortex-M CPUs. As good as the Cortex-M55 is, however, substantial additional performance uplift is revealed by the figures in the yellow columns: close to and even more than two orders of magnitude improvement when using the NPU+CPU compared to the CPU only. Noting that the Cortex-M55 already performs around 5x better than earlier, widely used Cortex-M architectures, it is reasonable to multiply these gains again by 5x. For speech recognition that would be around 800x faster and 400x less energy per inference compared to legacy Cortex-M CPUs.

Another important factor in tightly coupling the NPU to the CPU is the software development environment. There are many options for deploying proprietary and third-party NPU cores into SoCs. OEMs are very clear, however: they do not want to retool their entire development infrastructure to work with a new architecture just for AI/ML workloads, necessitating the adoption of new toolchains and new instruction sets.

If they already work in the Arm ecosystem for embedded control functions, they want to stay in the Arm ecosystem for AI/ML functions as well. Combining a Cortex-M CPU with an Ethos-U NPU, both from Arm, achieves this requirement.

In fact, the Ethos-U NPU is essentially a co-processor which integrates seamlessly with the Cortex-M CPU. The Arm Vela compiler will automatically split the ML workload between them, with 95% or more typically falling onto the NPU. A great side benefit is that the Cortex-M CPU can sleep or do other work while the ML inference is being resolved.

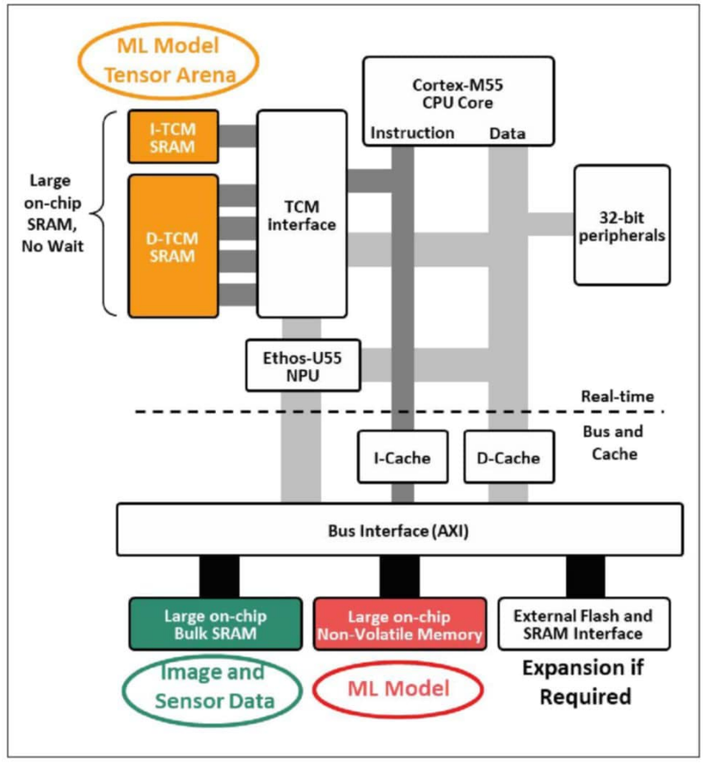


FIGURE 2
The Ensemble MCUs' internal memory topology

INTEGRATION HAS TO ENCOMPASS THE WHOLE SYSTEM

It's fair to say that the NPU is likely to be the center of attention in an integrated AI/ML MCU. However, what is integrated around the processing cores, and specifically how the integration is crafted, is very important. At the top of the list are memory and peripherals.

Figure 1 showed that enhanced processing capability is key to high performance and power efficiency, but without an optimized memory system behind it, the results will fail to meet expectations.

A simplified view of the Ensemble MCU memory topology is shown in **Figure 2**. The upper half represents the real-time section with very fast Tightly Coupled Memory (TCM) connected to the CPU and NPU cores. For fast inference times these TCM SRAM memories must be sufficiently large to hold the ML model's tensor arena.

The lower half of the diagram shows other system memories connected by a common high-speed bus. A large, shared bulk SRAM is required to hold sensor data, such as the input from a camera and microphones, and a large non-volatile memory contains the ML model itself plus application code. When large on-chip memories are distributed this way to minimize competing bus traffic, then concurrent memory transactions flourish, bottlenecks are cleared, memory access times are minimized, and power consumption is compatible with the use of a small battery.

The correct peripheral set is also critical for MCUs in endpoint ML applications which often operate in one or more of the 'three V' domains: vision, voice and vibration. This means that connectivity to image sensors, microphones, inertial measurement units and more is required, in addition to traditional MCU peripherals such as high-speed serial channels, analog interfaces, and display interconnects.

For endpoint AI devices, all these functions should be integrated into the MCU.

Whole-system integration not only eliminates the need for additional power rails and power conversion such as an external Power Management IC (PMIC), but also enables power to be controlled dynamically at a much more granular level on the chip – and this is the third desirable feature on an integrated AI MCU.

ADAPTIVE POWER MANAGEMENT TO EXTEND BATTERY RUNTIME

Alif recognized early on that the concentration of local ML capability at the endpoint will skyrocket in the near future, while at the same time the physical size of these products will rapidly shrink, especially for wearable devices, entailing the use of smaller and smaller batteries.

Alif's approach to stretching battery life to address this problem took several forms. Two prime examples are:

Partitioning the system so that a low-power portion of the chip can be always-on, but still offer robust compute capability, enabling it to selectively wake a much higher performance portion of the chip to execute heavy workloads then return to sleep

The power management system dynamically powers on only portions of the chip that are needed and shuts them off when not required, all at a finely granular level.

To facilitate this division of functions, many Ensemble MCUs have two pairs of Cortex-M55+Ethos-U55 cores as shown in **Figure 3**:

One in the High-Efficiency region of the chip built on low-leakage transistors that can be always-on operating at up to 160MHz. The other pair in the High-Performance region operating at up to 400MHz.

To picture the advantage this brings, imagine a smart occupancy camera which continuously scans a room at a low frame rate using the High-Efficiency pair of cores to classify a valid event (such as a human falling to the floor, or a specific gesture) which wakes the High-Performance pair to identify a person or persons, check for blocked exits, dial for help, and so on.

In this case the camera can be intelligently vigilant, produce fewer false positives, and extend battery life. Similar uses for these two pairs of CPU+NPU cores can be applied just as well to the classification of sounds, voices, words, text, vibrations, and sensor data in a wide variety of applications.

Additionally, all Ensemble MCUs employ Alif's aiPM™ (autonomous intelligent Power Management) technology to manipulate in real time up to 12 individual power domains in the chip as needed to match the use case being executed. Only domains that are actively executing tasks are powered on (such as those supplying specific processing cores, memories, or peripherals) while the other domains remain off. This becomes transparent to the software developer.

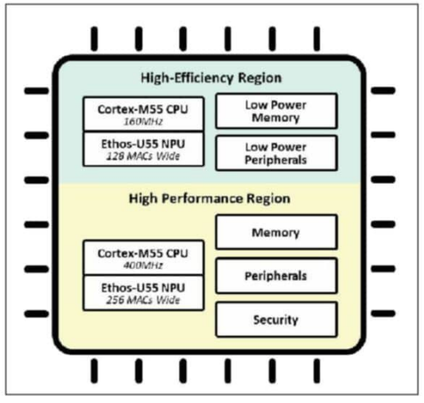


FIGURE 3
Ensemble E3 MCU block diagram showing the High-Efficiency and High-Performance regions of the chip.

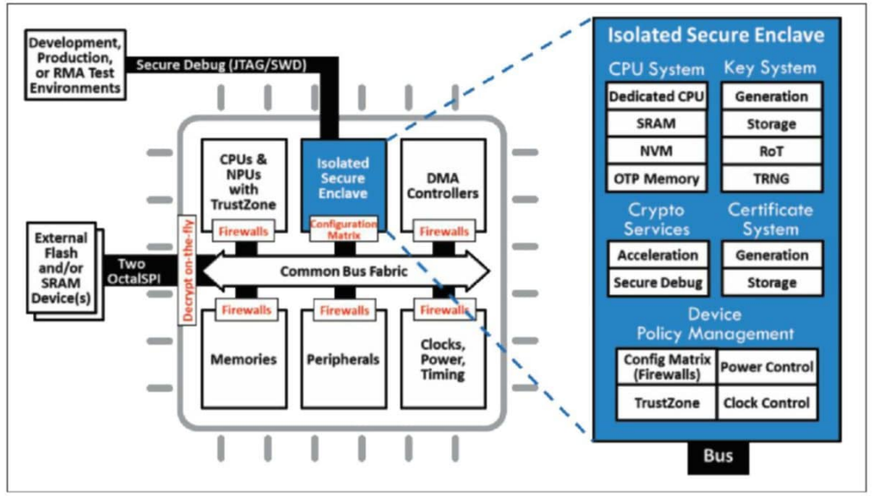


FIGURE 4
The secure enclave in Ensemble MCUs govern the security policy for the entire chip.

PROTECTION FOR VALUABLE MACHINE LEARNING MODELS AND OTHER IP

The final key feature which needs to be integrated into an AI MCU for endpoints is security protection. This is of course to resist the ever-present forms of cyber-attack. But for many OEMs the most important protection is for its IP embedded in AI models.

OEMs invest vast amounts of time and money in collating training datasets, building AI models, and developing and refining inferencing algorithms. This gives unscrupulous manufacturers a strong incentive to steal this expensive IP by copying it from insufficiently protected production units.

An external secure MCU would enable the OEM to establish a root-of-trust, manage secret keys and certificates, facilitate a secure boot, and so on. The use of an external secure MCU is a common approach to building in strong security to conventional MCU-based designs, but it is rare to find a full secure 'enclave' with these functions, and more, built into a conventional MCU.

Yet battery-powered and wearable AI products benefit particularly from the space and power savings, and the heightened security when this functionality is integrated into the MCU. The secure enclave (see **Figure 4**), standard in all Alif devices, is a dedicated isolated subsystem for management of vital security functions such as secure key management and storage, secure boot with an immutable Root-of-Trust, attestation at runtime using certificates, hardware cryptographic services, secure debugging, read-out protection, secure firmware updates, and complete lifecycle management.

AN AI-READY MCU PLATFORM

These four characteristics of an AI MCU – tight coupling of the NPU and CPU with a standard development ecosystem, whole-system integration, adaptive power management, and built-in IP protection – are in strong demand from the manufacturers of battery-powered endpoint devices that Alif Semiconductor has engaged with.

Designers who evaluate the Ensemble family will find a wide selection of scalable and compatible devices ranging from a single CPU core to quad-core devices supporting the Linux® operating system to fit varied projects while enabling re-use of software across all of them.

Learn more at www.alifsemi.com.

TECH THE FUTURE

RESOURCES
www.alifsemi.com

ABOUT THE AUTHOR

Mark Rootz is a successful business and technology executive with experience specializing in semiconductor product definition, productization, go-to-market, business development, and team building. Previously executing these functions at STMicroelectronics, Freescale, and Renesas, Mark continues this path as VP of Marketing at Alif Semiconductor.

19843342

SeaFire® Micros, Inc.
IP & Technology for Cloud Networks
Architecture
Protocol
SW FW i/f
hardware
L1-L7
TV
Phone
Data Center
Web
WiFi
Servers
PC Users
Design through Production
Military — Commercial
Hardware (Analog/Digital)
Firmware • Software • FPGA
Patented Network Technology | Unique Cloud Technology
No Compete Government Procurement
No Compete Government Funding
20% DISCOUNT on 1st Purchase
info@seafire.com | seafire.com

TS-9370

Low Power, High Performance SBC

The TS-9370 is a versatile and power-efficient industrial SBC built for rugged applications. Powered by the NXP i.MX9352 64-bit ARM® architecture plus two Cortex®-A53 processors operating up to 1.7 GHz, it delivers robust performance with low power consumption.

32 GB eMMC (upgradable)

2 GByte LPDDR4 RAM

MikroBUS socket
For MikroE Click Boards

Made in the USA
with global parts

VISIT US THIS MAY
IN HOUSTON, TX AT
OTC 2025
BOOTH #1675
XPONENTIAL
BOOTH #3725

embeddedTS

TS-4100

Ultra-Efficient and Fully Featured

The TS-4100 features an NXP i.MX 6UltraLite CPU, industrial temp range, FPGA, eMMC, and customizable I/O. Comes preloaded with Debian Stretch.

TS-7100-Z

As Small as Two Decks of Cards

Powered by an Arm® Cortex®-A7 i.MX6ul CPU, it runs efficiently at <2W, featuring Dual Ethernet, Bluetooth, WiFi, and NimbeLink/Digi sockets.

www.embeddedTS.com